目　　录

图形编程基础篇

图像处理技术篇

综合应用篇

图形编程基础篇

第一章　MATLAB 的句柄图形对象

句柄图形(Handle Graphics)是 MATLAB 中用于创建图形的面向对象的图形系统。图形句柄提供了多种用于创建线条、文本、网格和多边形等的绘图命令，以及交互式图形界面，如菜单、按钮和对话框等。

利用图形句柄，用户可以直接操纵线条、表面和其它图形对象。MATLAB 中的绝大多数图形函数和高级绘图命令正是利用这些图形元素来产生各种类型的图形的。利用图形句柄，我们可以在 MATLAB 的命令行中修改图形的显示效果，也可以在 M.文件中创建用户自定义的绘图函数。

本章主要内容：

★ MATLAB 的图形对象　　★ 图形对象属性值的设置和查询
★ 图形对象的属性　　★ 图形对象句柄的访问

1.1　MATLAB 的图形对象

图形句柄对象是 MATLAB 用来显示数据和创建图形用户界面(GUI)的基本绘图元素。对象的每个实例(instance)均对应于一个惟一的标识符，即句柄(handle)。利用该句柄，用户就可以非常容易地操纵现有图形对象的各种特征(即对象属性)。

MATLAB 中，对象的从属关系如图 1.1 所示。

从图 1.1 可以看出，所有 MATLAB 的图形对象都是彼此联系的。因此，典型的 MATLAB 图形显示均包含多种对象，而不是单一的。

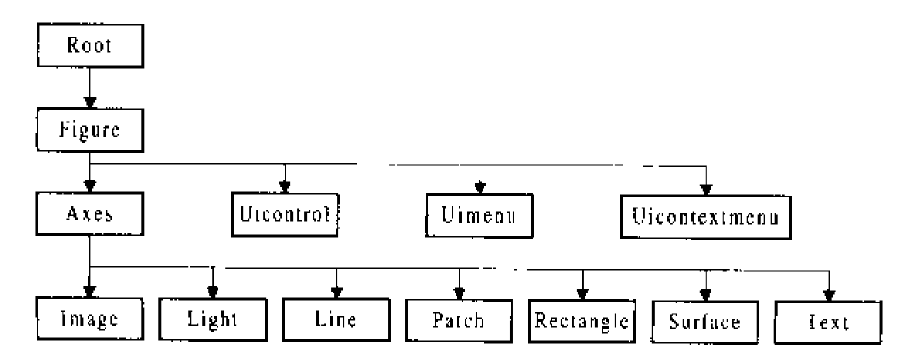


图 1.1　MATLAB 的对象从属关系图

MATLAB 遵循面向对象的原则，每个图形对象都有一个相应的构造函数，用于创建该对象类的实例。对象构造函数的名称与其构造的对象名相同。例如，text 函数创建 Text 对象(即文本对象)的实例；figure 函数创建 Figure 对象的实例等等。

下面的各小节将对图 1.1 中的各个对象进行详细的说明。

1.1.1 Root 对象

Root 对象即根对象。它位于 MATLAB 的对象继承图的最上层，因此在 MATLAB 中创建图形对象时，只能创建惟一的一个 Root 对象，而其它的所有的对象都从属于该对象。

需要注意的是，根对象是由系统在启动 MATLAB 时自动创建的，而用户可以对根对象的属性进行设置，从而影响图形的显示效果。

1.1.2 Figure 对象

Figure 对象即图形窗口对象，是在根屏幕(即 Root 屏幕)上显示 MATLAB 图形的独立窗口。用户可以在系统允许的前提下，在 MATLAB 中创建任意多个图形窗口。所有的图形窗口都是根对象的子对象，而所有其它的图形对象又是图形窗口对象(即 Figure 对象)的子对象。

如果当前还没有创建图形对象(即 Figure 窗口)，则调用任意一个绘图函数或图像显示函数(如 plot 函数、imshow 函数等)均可以自动创建一个图形窗口。如果当前的根对象已经包含了多个图形窗口，则总有一个窗口被指定为“当前”窗口，且该窗口为所有绘图函数的输出窗口。

在 MATLAB 中，用 figure 构造函数创建图形窗口的格式如下：

窗口句柄=figure('属性 1'，属性值 1，'属性 2'，属性值 2，…)

figure(窗口句柄)

其中，第一种格式既产生了一个窗口句柄，同时又对窗口对象的各个属性进行了具体的设置，且按照该设置生成了一个图形窗口

例 1.1.1 创建图形窗口。

```
h=figure ('color',[1 1 1],'position',[0 0 200 100],...
          'name','image processing','NumberTitle','on',...
          'menu','figure','WindowButtonDown',...
          'disp("welcome to this program")');
figure(h);
```

结果产生如图 1.2 所示的窗口。

另外，对于第二种格式，即 figure(窗口句柄)格式，如果窗口句柄是存在的，则将调用该窗口并使其处于可视状态且设定为“当前”窗口；如果窗口句柄不存在，则该函数将针对这一句柄生成一个新的窗口，并设定为“当前”窗口。

例如，执行下面的语句：

figure(h);

将以系统默认的设置生成一个图形窗口，如图 1.3 所示。

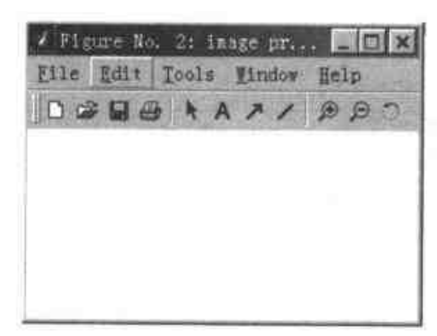


图 1.2 用户设置的图形窗口



图 1.3 系统默认的图形窗口

另外，对图形窗口属性的设置还可调用 set 函数，其格式如下：

set(窗口句柄，'属性 1'，属性值 1，'属性 2'，属性值 2，…)

要注意的是，在调用 set 函数之前，该窗口句柄应该是已经存在的。

图形窗口的各种常用属性的设置及其可设置的属性值如表 1.1 所示。

表 1.1 图形窗口的常用属性与属性值

属 性	说 明
Color	图形的背景颜色，可设置为三元素的 RGB 向量或者是 MATLAB 自定义的颜色名，MATLAB 的缺省值为黑色。RGB 向量分量的取值范围为[0 1]，取极端值可得到以下八种颜色：[0 0 0] 黑色，[1 1 1] 白色，[0 0 1] 蓝色，[0 1 0] 绿色，[0 1 1] 天蓝色，[1 0 0] 红色，[1 0 1] 粉红色，[1 1 0] 黄色
CurrentAxes	当前图形坐标轴的句柄
CurrentMenu	最近被选择的菜单项的句柄
CurrentObject	图形内最近被选择的对象的句柄。可由 gco()函数获得
MenuBar	设置图形窗口菜单条形式，'figure'显示缺省的 MATLAB 菜单，'none'为不加菜单。在选择'none'后，可以用 uimenu()函数设置自定义菜单；在选择'figure'后，还可以用 uimenu()函数添加新菜单
Name	设置图形窗口的标题栏内容，它的属性为一个字符串，在创建窗口时，字符串将添入标题栏
NameTitle	设置图形窗口标题栏图形标号的有无，取'on'将自动地给图形窗口标题前加上 Figure No 字样；取'off '则将不再给标题前加标号
Position	位置向量[left, bottom, width, height]，(left, bottom)代表窗口左下角的坐标，width 和 height 分别表示窗口的宽度和高度
Uints	设置尺寸的单位。'inches'，即 in；'centimeters'，即 cm；'normalized'为归一化坐标，取值范围为[0 1]；'point'为排字点，取值为 0.353 mm；pixel 为屏幕像素，这是 MATLAB 默认的单位
Resize	设置是否可以用鼠标调整窗口的大小，'on'为可调；'off '为不可调
Pointer	设置窗口下指示鼠标光标的显示形式，'crosshair'表示十字型；'arrow'表示箭头；'watch'表示沙漏；'topl'指向左上方的箭头；'topr'指向右上方的箭头；'bopl'指向左下方的箭头；'bopr'指向右下方的箭头；'circle'表示圆形光标；'cross'表示双线十字；'fleur '表示带有箭头的十字花形
Visible	设置窗口初始时刻是否可见，选项有'on'和'off '，其中 MATLAB 的缺省值为'on'，在编程的过程当中，不必看见中间的过程，可以首先使用'off '，在完成设置后，再选取'on'来显示窗口
PaperSize	设置打印纸张的大小。向量[width, height]代表了打印纸张的宽度和高度

续表

属　性	说　　明
PaperType	设置打印纸张的类型。'usletter'为标准美国信纸；'uslegal'为标准美国法定纸张；'taboid'为标准美国报纸；'a3'为A3纸；'a5'为A5纸；'b4'为B4纸；'b5'为B5纸
PaperUints	设置纸张属性的度量单位。'inches'，即in；'centimeters'，即cm；'normalized'表示归一化坐标，取值范围为[0 1]；'points'即像素点，取值为0.353 mm
PaperOrientâtions	设置打印时的纸张方向。'portrait'为纵向打印，这是MATLAB的缺省值；'landscape'为横向打印
PaperPosition	设置打印页面上的图形位置，位置向量[left, bottom, width, height]，(left, bottom)代表打印页面图形左下角的坐标，width和height分别表示打印页面图形的宽度和高度

1.1.3 Uicontrol 对象

Uicontrol对象即控制框对象，是图形用户界面(GUI)控件，当用户激活该对象时，系统将执行相应的回调函数。Uicontrol对象的控件有多种类型，包括按钮、无线电按钮、检查框、编辑框、静态文本框、列表框、滑标等。一个良好的用户界面是与这些控件分不开的。

表1.2给出了控制框的控件类型及相应的属性值。

表 1.2　控制框的控件类型

控　件	说　　明
按钮 (button)	按钮是对话框中最常用的控制图标，按钮上通常有提示的文本，其通常表示执行一个动作，而不是用来改变对象的状态和属性。其'style'属性值为'pushbutton'
无线电按钮 (radio)	无线电按钮是一组带有文字提示的选择项，通常用于一组互斥的选项当中，每次只能选一项。当一项被选中后，被选中按钮的圆中心显示为黑色。其'style'属性值为'radiobutton'
检查框 (checkbox)	检查框通常是由具有标志的一个小方框组成的，其作用与无线电按钮极相似，主要区别是检查框可同时选择多个选项。其'style'属性值为'checkbox'
编辑框 (editbox)	编辑框是一个含有初始值的方框或无初值的空框。用户可以在里面输入自己的数据，相关的程序将会获取框中的数据信息。其'style'属性值为'checkbox'。
静态文本框 (statictext)	静态文本框的主要作用是显示标志、用户提示信息和一些特定数据的当前值。其'style'属性值为'edit'
滑标 (slider)	滑标又称滚动条，它是由滚动槽、长方条区域、指示器三部分组成的。其中，长方条区域代表有效值范围，指示器代表滑标的当前值。其'style'属性值为'slider'
列表框 (listbox)	列表框用来列出可以选择的所有选项，用户可以方便地从中选择任意选项，如果选项太多，可采用垂直滚动条加以控制。其'style'属性值为'listbox'
弹出式菜单 (popupmenu)	弹出式菜单通常用以向用户提出一系列互斥选项的清单，以便用户选择，弹出式菜单不受菜单条的限制，可以出现在窗口的任何位置。其'style'属性为'popmenu'

用户可以将Uicontrol对象控件组合使用，以构建控制面板和对话框。

控制框的创建用uicontrol()函数。该函数用法如下：

控制框句柄=uicontrol(父对象的句柄，'属性1'，属性值1，'属性2'，属性值2，...)

表1.3给出了Uicontrol对象的一般属性。

表 1.3　Uicontrol 对象的一般属性

属　性	说　明
ForegroundColor	设置对象的前景颜色
BackgroundColor	设置对象的背景颜色
Callback	设置回调字符串，当控制框被选中时，执行响应的函数
HorizontalAlignment	设置当前控件在水平方向的对齐方式，可选的值有'left'、'center'、'right'，分别代表左、中、右的对齐方式，'right'为默认值。该属性对于单个控件不起作用，只有在多个控件共存时才起作用
Max	设置属性'value'的最大有效值，且该最大值取决于 Uicontrol 对象的'type'属性。当控制框对象无线电按钮和检查框处于'on'状态时，将 value 设定为 max；它设定了弹出式菜单最大下标值和滑标的最大下标值。当 max - min>1 时，编辑文本框为多行文本。其默认值为 1
Min	设置属性'value'的最小有效值，且该最小值取决于 Uicontrol 对象的'type'属性。当控制框对象无线电按钮和检查框处于'off'状态时，将 value 设定为 min；它设定了弹出式菜单最小下标值和滑标的最小下标值。当 max - min>1 时，编辑文本框为多行文本。其默认值为 0
Position	设置控件的位置属性，用位置向量[left bottom width height]来表示
String	设置控件上的字符串，在按钮、无线电按钮、检查框和弹出式菜单上的显示标志
Style	设置控件类型，细节请参照表 1.2
Type	设置可读对象的辨识串，通常是用户指定的数据
Value	设置 uicontrol 的当前值。无线电按钮和检查框处于'on'状态时，将 value 设定为 max；无线电按钮和检查框处于'off'状态时，将 value 设定为 min。滑标把 value 设置为 min≤value≤max，弹出式菜单则把 value 设置为 1≤value≤max
Visible	设置对象的可视状态，'on'为可视；'off'为不可视

下面举一个创建控制框对象的简单例子，更为详细的过程请参照本书的最后一部分——用户界面(GUI)的设计。

例 1.1.2　创建控制框对象。

```
H=figure('Color',[0,1,1],'Position',[0,0,400,400],'Name',...
   '关于控制框设计的程序','NumberTitle','off',...
   'MenuBar','none');
uicontrol(H,'Style','text','Position',[0.05,0.74,0.9,0.1],...
      'Uints','normalized','String','WELCOME TO THIS...
PROGRAM!','Back',[0,1,1].'Fore',[1,0,0]);
uicontrol(H,'Style','frame','Position', ...
[0.15,0.35,0.5,0.40],'Uints','normalized','Back',[1,1,0]);
uicontrol(H,'Style','edit','Position', ...
```

```
[0.42,0.65,0.2,0.06],...
    'Uints','normalized','String','256','Back',[0,1,0]);
uicontrol(H,'Style','slide','Position', ...
 [0.05,0.05,0.9,0.08],...
    'Uints','normalized','Back',[1,1,0],'max',1,'min',0);
radio=uicontrol(H,'Style','radio','Position', ...
[0.2,0.65,0.2,0.06],...
    'Uints','normalized','String','输入N','Back',[1,1,0]);
checkbox(1)=uicontrol(H,'Style','checkbox','Position', ...
[0.2,0.54,0.2,0.06],...
    'Uints','normalized','String','N=100','Back',[1,1,0]);
checkbox(2)=uicontrol(H,'Style','checkbox','Position', ...
[0.2,0.43,0.2,0.06],...
    'Uints','normalized','String','N=200','Back',[1,1,0]);
change=['disp(''改变窗口的颜色'')'];
uicontrol(H,'Style','push','Position',[0.7,0.35,0.2,0.06],...
    'Uints','normalized','String','cancel','Call','close(H)');
uicontrol(H,'Style','push','Position', ...
[0.05,0.17,0.85,0.06],...
    'Uints','normalized','String','CHANGE WINDOWS ...
COLOR','Call',change);
```

执行的结果如图 1.4 所示。

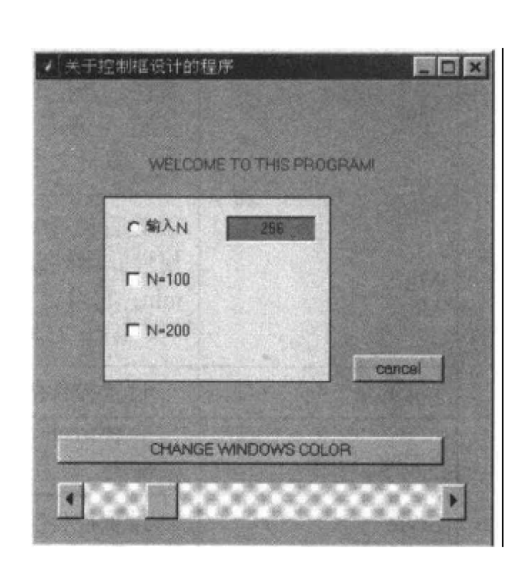


图 1.4 控制框的设计

1.1.4 Uimenu 对象

在一个方便简洁的用户图形界面设计中，菜单的设计是至关重要的。有了一个完备的菜单，用户就可以方便地从菜单中选择各条命令和各个选项。MATLAB 为图形用户界面提供了一个标准的菜单，其中包括四个下拉式菜单：File 菜单、Edit 菜单、Windows 菜单和 Help 菜单。

除了上面已经提到的标准菜单外，MATLAB 还提供了 uimenu()构造函数来创建 Uimenu 对象(即下拉式菜单对象)，从而设计用户感兴趣的个性化菜单。

uimenu()函数的用法如下：

菜单项句柄=uimenu(窗口句柄，'属性 1'，属性值 1，'属性 2'，属性值 2，…)

子菜单句柄=uimenu(菜单项句柄，'属性 1'，属性值 1，'属性 2'，属性值 2，…)

菜单对象可以设置的属性如表 1.4 所示。

Uimenu 对象属性中，最重要的是'Label'和'Callback'属性。'Label'属性决定了菜单的名称，'Callback'属性决定了菜单项的响应，如果不设置'Callback'属性，菜单项将失去意义。

表 1.4 菜单对象的常用属性

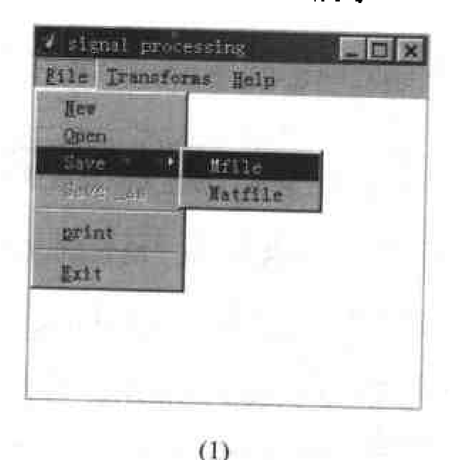
属性	说明
ForegroundColor	设置菜单的前景色，可以设置为 RGB 向量或 MATLAB 自定义的颜色名称
BackgroundColor	设置菜单的背景色，可设置为 RGB 向量或 MATLAB 自定义的颜色名称，默认值为黑色
Callback	设置 MATLAB 的回调字符串，当该菜单被选中时，回调字符串传给函数 eval()，其初值为空矩阵
Checked	设置被选项的校验标记，如果为'on'，则校验标记出现在所选菜单的旁边；若为'off '，则不出现校验标记
Enable	设置菜单使能状态。若为'on'，则可用，当选中该菜单时执行回调函数；若为'off '，则菜单项不可用，菜单标志变灰
Label	设置菜单条名称，通常是一个字符串，在菜单项字符中可使用'&'标志以表示该符号后面的字符在显示时有一个下划线修饰，以便用户可以使用键盘 Alt+该字符来操作菜单
Position	设置菜单项的相对排列位置，可以用来测试菜单项处于整个菜单系统的第几级
Separator	设置菜单项的分界符，'on'为在该菜单项的上面加上一条横线作为分界符，'off '为不加分界符，MATLAB 的默认值为'off '
Parent	父对象的句柄，如果 uimenu 对象是顶层菜单，则其为图形对象；如果 uimenu 对象是子菜单，则其为父菜单的句柄值
Visible	设置 uimenu 对象在屏幕上的可见形，'on'为可见，'off '为不可见

例 1.1.3 创建菜单对象。

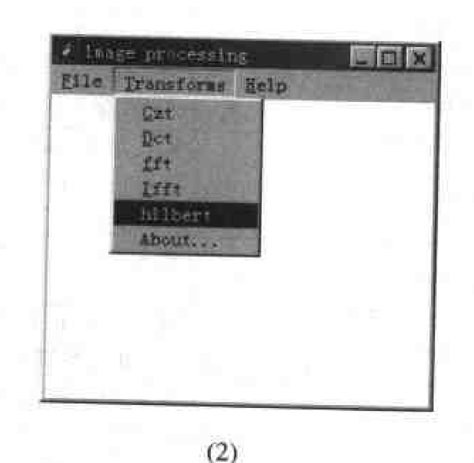
```
H= figure('Color',[1 1 1],'position',[0,0,300,400],...
    'Name','signal processing','NumberTitle','off ',...
    'MenuBar','none');
mfile=uimenu(H,'label','&File');
mtran=uimenu(H,'label','&Transforms');
mhelp=uimenu(H,'label','&Help');
uimenu(mfile,'label','&New','call',...
      'disp(''new file selected'')');
uimenu(mfile,'label','&Open','call',...
      'disp(''open file selected'')');
msave=uimenu(mfile,'label','Save','Enable','on');
uimenu(msave,'label','Mfile','call','savem.m');
uimenu(msave,'label','Matfile','call','savemat.m');
uimenu(mfile,'label','Save & As','Enable','off');
uimenu(mfile,'label','&print','separator','on', ...
      'call','print.dwin.loose.f ');
uimenu(mfile,'label','&Exit','separator','on', ...
      'call','close(H)');
uimenu(mtran,'label','&Czt','call','help czt');
```

```
uimenu(mtran,'label','&Dct','call','help dct');
uimenu(mtran,'label','&fft','call','help fft');
uimenu(mtran,'label','&Ifft','call','help ifft');
uimenu(mtran,'label','hilbert','call','help hilbert');
uimenu(mtran,'label','About...','call',...
        'disp(''This is a menu.making example'')');
```

代码执行的结果如图 1.5 所示。



(1)



(2)

图 1.5 用 uimenu 函数创建的菜单

这只是一个简单的创建菜单的例子，关于如何创建一个功能更加完善的菜单，请参照本书的最后一部分——用户界面(即 GUI)的设计。

1.1.5 Axes 对象

Axes 对象即坐标轴对象。它在图形窗口中定义一个区域，并将其所有子对象都限制在该区域内。坐标轴对象是图形窗口对象(Figure 对象)的子对象，同时还是 Image 对象(图像对象)、Light 对象(灯光对象)、Line 对象(线条对象)、Patch 对象(片块对象)、Surface 对象(表面对象)和 Text 对象(文本对象)的父对象。

如果当前运行的 MATLAB 输出窗口中还没有坐标轴对象，则任意一个绘图函数(如 plot 函数、surf 函数等)都可以创建坐标轴对象。如果在当前图形窗口中包含多个坐标轴对象，则总有一个坐标轴对象被指定为“当前”坐标轴，并且，该“当前”坐标轴即为所有绘图函数输出的坐标基准。

Axes 对象是由 axes()构造函数创建的。

axes 函数的调用格式如下：

坐标轴句柄=axes('position'，rect)

其中，rect=[left, bottom, width, height]定义了坐标系的位置和尺寸。单位归一化后，(0，0)即为当前图形窗口中左下角点的坐标，而(1.0，1.0)则为当前图形窗口中右上角点的坐标。

例如，执行下面的代码：

```
axes('position',[0.1,0.2,0.5,0.5]);
```

将在(0.1，0.2)处，创建一个宽度和高度均为 0.5 个单位的坐标系，结果如图 1.6 所示。

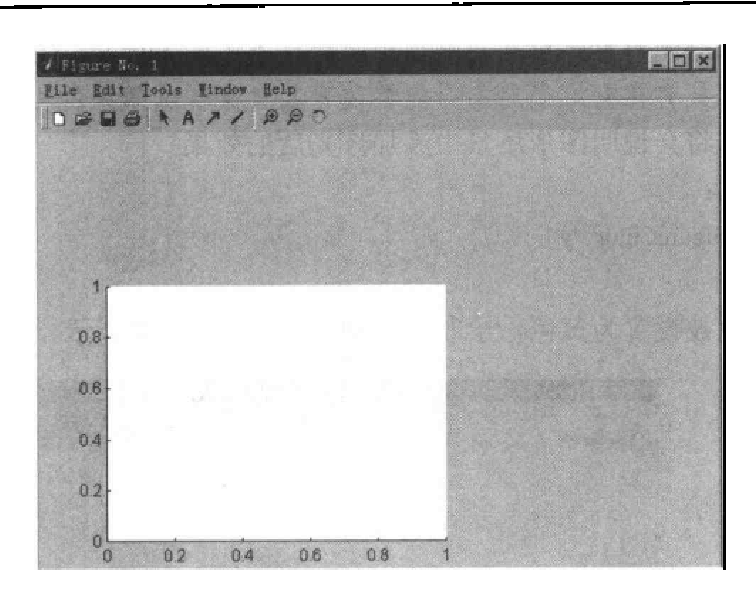


图 1.6 用 axes 构造函数创建的坐标系

另外，在 MATLAB 中，与坐标轴对象相关的函数还有如下几个：

(1) axis——该命令用于设置影响当前坐标轴对象的相关值。如执行下面的命令：

axis([xmin, xmax, ymin, ymax])

可以设置当前坐标轴的坐标值范围。

(2) get——该命令用于获取现有的坐标轴的各项属性值。

(3) set——该命令用于设置现有的坐标轴的各项属性值。

(4) gca——该命令用于返回当前坐标轴的句柄(即标识号 ID)。

1.1.6 Image 对象

Image 对象即图像对象。它包含一个数据矩阵，有时还包含一个颜色映像表。根据数据矩阵元素代表意义的不同，MATLAB 中的 Image 对象有三种基本的类型，即索引图像、灰度图像和真彩图像。

Image 对象是由构造函数 image()来创建的，其具体用法可参见第三章。

1.1.7 Light 对象

灯光设置是 MATLAB 中为增加图形场景的视觉效果而提供的一种处理技术。该项技术是通过模拟对象实体在自然光下的明暗特征来实现的。在 MATLAB 中，如果希望产生灯光效果，必须创建一个“灯光”对象。

MATLAB 中利用 light()构造函数来创建灯光对象。每个灯光对象都包含三个非常重要的属性：

(1) Color——该属性用于指定来自于光源方向的灯光颜色。其可以影响到图形场景中对象的显示颜色。

(2) Style——该属性用于指定光源类型是点光源还是无限远光源。当将 Style 设置为 local 时，表示是点光源；设置为 infinite 时，表示是无限远光源。默认值为后者。

(3) Position——该属性用于指定光线方向(对于无限远光源)或光源位置(对于点光源)。

需要注意的是，灯光对象只影响与其在相同坐标系下的 Surface 对象和 Patch 对象。因为只有这两种对象才具有在有灯光对象时能够改变其自身视觉效果的相关属性。

下面这个例子，将会说明图形场景中添加灯光后的效果。

执行下面的代码：

```
set(0,'DefaultFigureColor', 'y')
membrane
```

图形窗口的颜色被设置为黄色，绘制出的表面图形如图 1.7 所示。

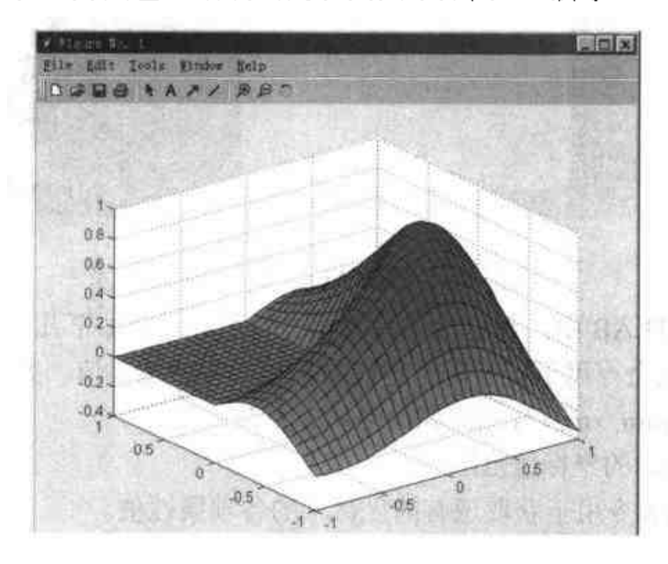


图 1.7　没有灯光的表面图形

添加灯光的代码如下：

```
light('position',[0 -1 1],'color','r')
```

该光源为无限远光源，其中矢量[0 -1 1]定义了光源的方向，即该光源顺序通过坐标原点(0，0，0)和点(0，-1，1)；而且还定义了该光源的颜色为红色，用'r'表示。最终的效果如图 1.8 所示。

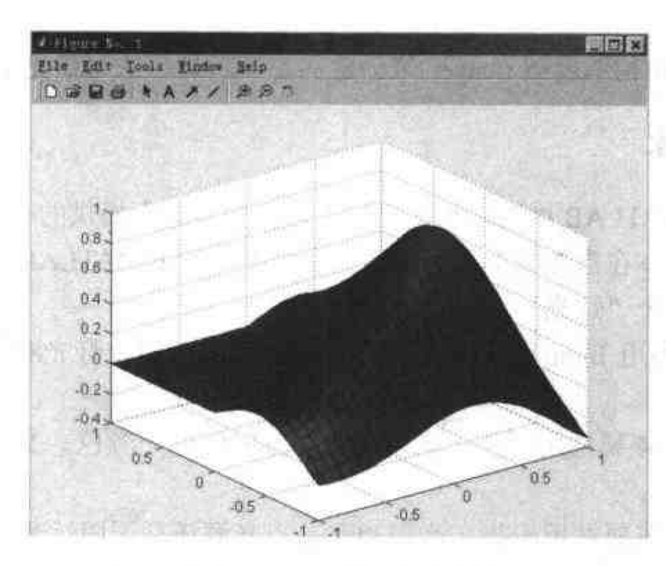


图 1.8　添加了灯光效果后的表面图形

1.1.8 Line 对象

Line 对象即线条对象。它是最基本的图形对象，主要用于创建绝大多数的二维线条和某些三维线条。线条对象的父坐标系负责线条的定位。

MATLAB 中利用 line()构造函数来创建线条对象。

line()函数的基本调用格式如下：

线条句柄=line(x,y)

其中，x 和 y 既可以是向量，也可以是矩阵。如果都是向量，则只绘制一个线条；如果是大小相同的矩阵，则对应的每一列都创建一个线条。

例 1.1.4 创建线条对象。

```
x=[0.5,1.2,2;-0.5,0,1;1,2,3];
y=[0,1,2.5;0.5,1,1.5;-1,0.2,1.3];
line(x,y);
```

由于 x 和 y 都是 3×3 的矩阵，即都包含 3 列，因而创建了 3 个线条，如图 1.9 所示。

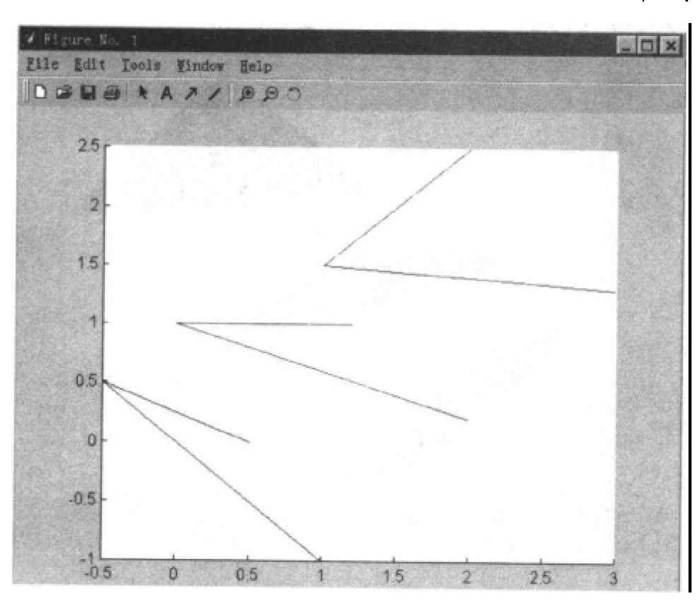


图 1.9 用 line 函数创建的线条对象

此外，MATLAB 中的许多高级函数(例如 plot 函数、plot3 函数和 loglog 函数等)也都可以创建线条对象。

1.1.9 Patch 对象

Patch 对象即片块对象。它是由一个或多个相连或不相连的多边形组成的。在 MATLAB 中，在对实物对象(例如飞机场、汽车等)进行建模，或绘制任意形状的二维或三维多边形时，经常会用到片块对象。

单个片块可以包含多个面，每个面均有特定的颜色。片块对象的父坐标系负责片块的定位。

MATLAB 中利用 patch()构造函数来创建片块对象。

patch()函数的基本调用格式如下：

patch(x,y,c)

其中，x 和 y 既可以是向量，也可以是矩阵。如果都是向量，则只绘制一个片块；如果是大小相同的矩阵，则连接 x 和 y 的每一列中的各个元素所组成的点(x,y)都构成一个片块。而 c 则表示填充各个片块的颜色，其可以是字符，如'r','g','b','c','m','y','w'和'k'，也可以是 1×3 的向量，表示 RGB 值，如[1 0 0]代表红色。

例 1.1.5 创建片块对象。

```
x=[0.5,1.2,2;-0.5,0,1;1,2,3];
y=[0,1,2.5;0.5,1,1.5;-1,0.2,1.3];
patch(x,y,[1 0 0]);
```

由于 x 和 y 都是 3×3 的矩阵，即都包含 3 列，因而创建了 3 个片块，且填充颜色都是红色，如图 1.10 所示。

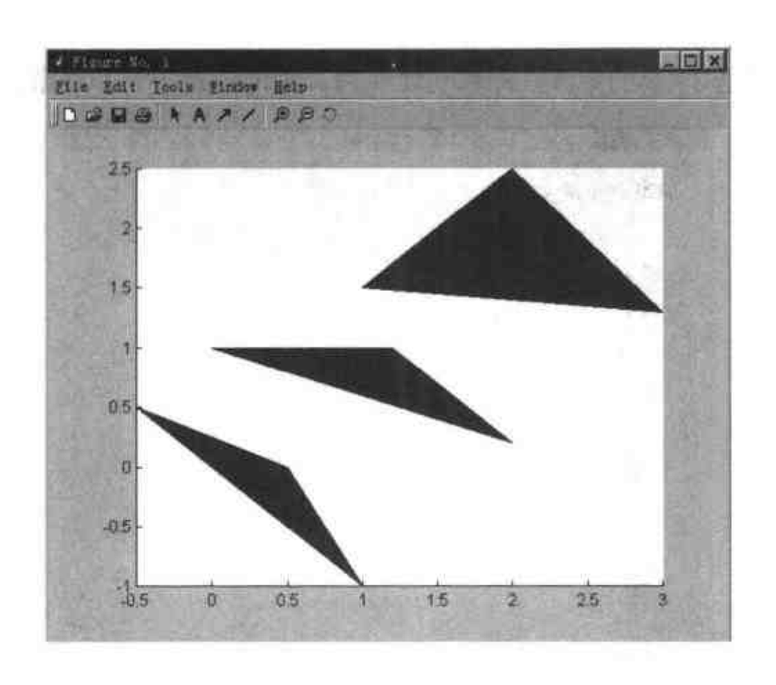


图 1.10 patch 函数创建的片块对象

此外，MATLAB 中的 fill 函数、fill3 函数和 contour 函数也都可以创建片块对象。

1.1.10 Rectangle 对象

Rectangle 对象即矩形对象。它是一个二维填充区域，其形状可以是矩形，也可以是平滑的矩形(rounded_rectangle)或椭圆。

MATLAB 中利用 rectangle()构造函数来创建矩形对象。

rectangle()函数的基本调用格式如下：

矩形对象的句柄=rectangle('position',[x y width height])

其中，(x,y)代表矩形左下角的坐标，width 代表矩形的宽度，height 代表矩形的高度。

另外，下面的代码产生的也是矩形：

```
rectangle('Curvature',[0 0])
```

下面的代码产生的是椭圆：

```
rectangle('Curvature',[1 1])
```

而下面的代码产生的则是平滑后的矩形：

```
rectangle('Curvature',[x y])
```

其中，x 和 y 取[0 1]之间的值，x 代表水平方向弯曲的程度，y 代表垂直方向弯曲的程度。

例如，下面的代码：

```
rectangle('Curvature', [0.1 0.3],'facecolor','b')
```

将创建一个平滑了的矩形，且用蓝色填充该矩形，结果如图 1.11 所示。

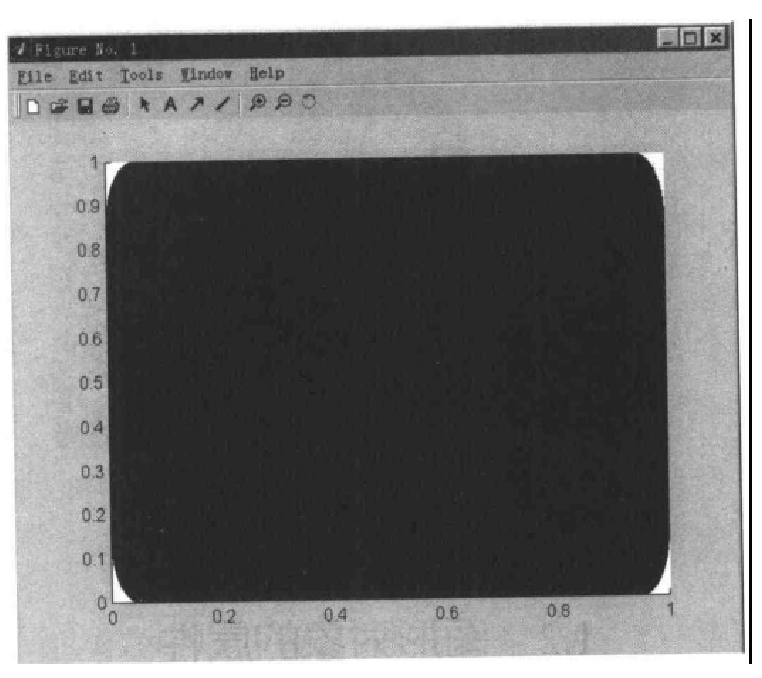


图 1.11 由 rectangle 函数创建的平滑了的矩形对象

1.1.11 Surface 对象

Surface 对象即表面对象。它是矩阵数据的三维表征。表面图形对象是由一系列的小四边形组成的，四边型的每个顶点的位置是由矩阵中的对应的元素确定的。

MATLAB 中利用 surface()构造函数来创建表面对象。

surface()函数的基本调用格式如下：

表面图形句柄=surface(x,y,z)

其中，x,y,z 为三个大小相同的矩阵，三个矩阵中对应的点(x,y,z)即为该表面对象各个小四边形顶点的坐标。

例 1.1.6 创建表面对象。

```
x=[1 2 3;4 6 7;1 3 9];
y=[0.1 0.3 0.5;2 3 0.2;4 6 8];
```

```
z=[0.1 0.3 0.5;2 3 0.2;4 6 8];
h=plot3(x,y,z);
delete(h);
surface(x,y,z);
```

创建了一个表面对象，结果如图 1.12 所示。

此外，MATLAB 中的高级函数 surf 和 pcolor，以及网格绘制函数均可以创建表面对象。

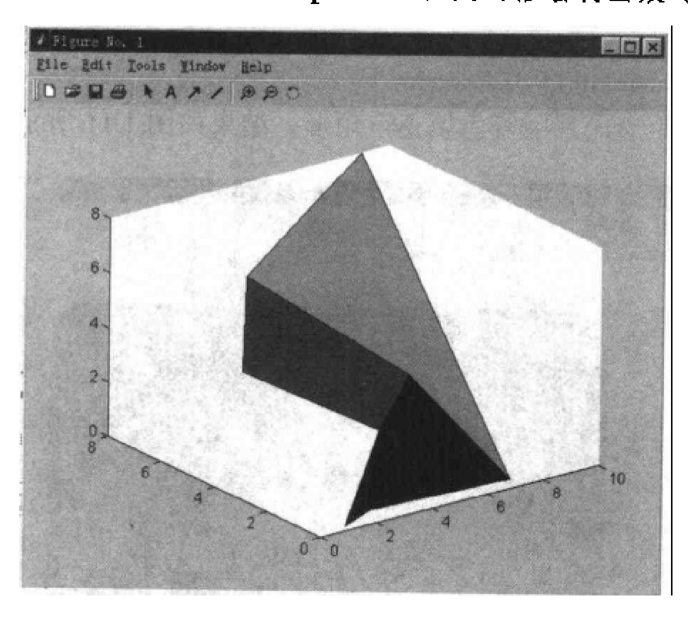


图 1.12 由 surface 函数创建的表面对象

1.2 图形对象的属性

图形对象的属性控制图形的外观和显示特点。图形对象的属性包含公共属性和特有属性。

MATLAB 将图形信息组织在一个有序的金字塔式的阶梯图表中，并将其存储在对象属性中。例如，根对象的属性包含当前图形窗口对象(Figure 对象)的句柄和鼠标指针的当前位置；而图形窗口对象属性则包含其子对象的列表，同时跟踪发生在当前图形窗口中的某些 Windows 事件；坐标轴对象属性则包含其每个子对象(图形对象)使用图形颜色映射表的信息和每个绘图函数对颜色的分配信息。

通常情况下，用户可以随时查询和修改绝大多数属性的当前值，而有一部分属性对用户来说是只读的，只能由 MATLAB 修改。需要注意的是，任何属性只对某个对象的某个具体的实例才有意义，所以，修改同一种对象的不同实例的相同属性时，彼此互不干涉。

用户可以为对象属性设置默认值，则此后创建的所有该对象的实例所对应的这个属性的值均为该默认值。

表 1.5 给出了 MATLAB 中图形对象的公共属性。

表1.5 MATLAB图形对象的公共属性

属　性	说　明
BusyAction	控制MATLAB图形对象句柄回调函数的中断方式
ButtonDownFcn	当单击按钮时执行回调函数
Children	该对象所有子对象的句柄
Clipping	使能或非使能剪切模式(只对坐标轴子对象有效)
CreateFcn	当同种类型的对象创建时执行回调函数
DeleteFcn	当用户发出删除该对象时执行回调函数
HandleVisibility	容许用户控制来自于命令行和回调函数内部的对象句柄是否可用
Interruptible	确定当前的回调函数是否可以被后继的回调函数中断
Parent	该对象的父对象
Selected	表明该对象是否被选中
SelectionHighlight	指定选中的对象是否以可视化的方式显示出来
Tag	用户指定的对象标签
Type	该对象的类型(是Figure对象、Line对象，还是Text对象等等)
UserData	用户想与该对象关联的任意数据
Visible	指明该对象是否可见

1.3 图形对象属性值的设置和查询

在创建MATLAB的图形对象时，通过向构造函数传递"属性名/属性值"参数对，用户可以为对象的任何属性(只读属性除外)设置特定的值。首先通过构造函数返回其创建的对象句柄，然后利用该句柄，用户可以在对象创建完成后对其属性值进行查询和修改。

在MATLAB中，set函数用于设置现有图形对象的属性值；get函数用于返回现有图形对象的属性值。利用这两个函数，还可以列出具有固定设置的属性的所有值。

下面具体介绍如何利用这两个函数来实现对图形对象属性值的设置和查询。

1.3.1 属性值的设置

MATLAB中，用户利用set函数和对象相应的构造函数的返回句柄，可以设置该对象的各项属性。

set函数的基本调用格式如下：

set(对象句柄，'属性1'，属性值1，'属性2'，属性值2，…)

例1.3.1 执行下面的代码：

```
t=0:pi/20:2*pi;
z=sin(t);
plot(t,z);
```

```
set(gca,'YAxisLocation','right');   %设置当前坐标轴的 y 轴的位置属性为右
xlabel('x');
ylabel('y');
```

将 y 轴移到当前坐标系的右边，结果如图 1.13 所示。

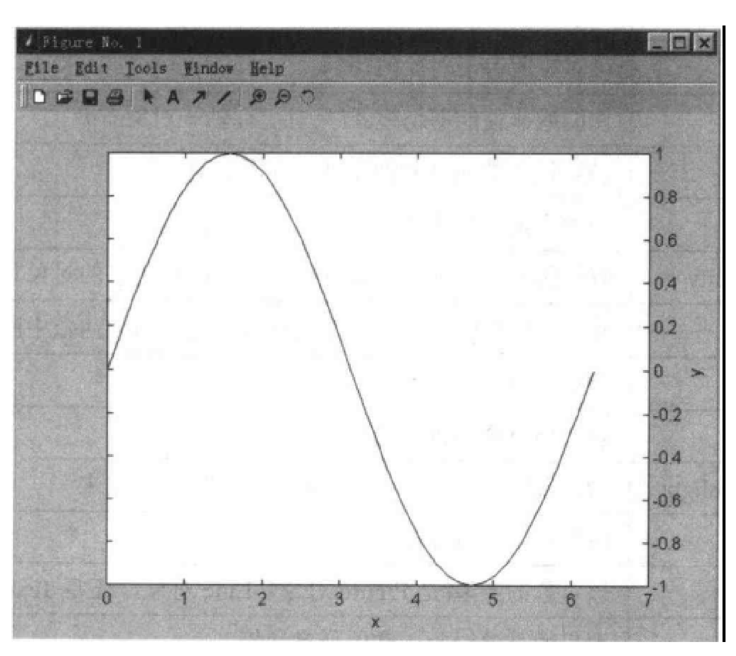


图 1.13　将 y 轴移到坐标系右边的图形显示

如果对象句柄是一个矢量，则 set 函数对该矢量中所有的对象均可以进行属性的设置。

另外，利用 set 函数可以得到多个属性的所有可能的值。例如，执行下面的代码将会得到用户可设定的 Line 对象(即线条对象)的线形的所有值：

```
set(line,'LineStyle')
```

得到 Line 对象的属性 LineStyle 的属性值列表如下：

```
[ {.} | .. | : | .. | none ]
```

其中，大括号内的值为默认值。

如果想查看一个对象，如 Line 对象的所有可设置属性(即非只读属性)的所有可能的值，则使用如下的格式：

```
set(line)
```

得到如下的结果：

```
Color
EraseMode: [ {normal} | background | xor | none ]
LineStyle: [ {.} | .. | : | .. | none ]
LineWidth
Marker: [ + | o | * | . | x | square | diamond | v | ^ | > | < | pentagram | hexagram | {none} ]
MarkerSize
MarkerEdgeColor: [ none | {auto} ] .or. a ColorSpec.
MarkerFaceColor: [ {none} | auto ] .or. a ColorSpec.
```

XData

YData

Zdata

⋮

UserData

Visible: [{on} | off]

1.3.2　对象的默认属性值

在 MATLAB 中，所有的对象属性均有系统默认的属性值，即工厂(factory)设定值。同时，用户也可以自己定义任何一个 MATLAB 对象的默认属性值。

1. 默认属性值的搜索

MATLAB 对默认属性值的搜索从当前对象开始，沿着对象的从属关系图向更高的层次搜索，直到发现系统的默认值或用户自己定义的值。

定义对象的默认值时，在对象从属关系图中，该对象越靠近 Root 根对象，其作用的范围就越广。例如，如果我们在根对象的层次上为 Line 对象定义了一个默认值，由于根对象位于对象从属关系图的最上层，因此该值将会作用于所有的线条对象。

如果用户在对象从属关系图的不同层次上定义同一个属性的默认值，则 MATLAB 将会自动选择最下层的属性值作为最终的属性值。需要注意的是，用户自定义的属性值只能影响到该属性值设置后创建的对象，之前和现有的对象都不受影响。

2. 默认属性值的设置

指定 MATLAB 对象的默认值，要首先创建一个以“Default”开头的字符串，该字符串的中间部分为对象类型，最末尾为对象的属性名称。

例如，指定当前图形窗口对象层上线条对象的线宽属性的默认值的代码如下：

```
set(gcf,'DefaultLineLineWidth',2.5)
```

其中，字符串 DefaultLineLineWidth 标识了所要设置默认值的属性为线条对象的线宽属性。而句柄 gcf 则指示了所设置属性值所在的对象层，此处即为当前图形窗口对象层。

执行完上述代码后，再执行下面的代码：

```
line
```

得到如图 1.14 所示的结果。

又如下面的代码：

```
set(0,'DefaultFigureColor', 'w')
```

使用了字符串 DefaultFigureColor，用于指定图形窗口的默认的颜色。此处的句柄参数为 0，表示是在根对象层指定图形窗口的默认的颜色。

另外，利用 get 函数可以得到当前某一对象层上所有默认的设置。例如执行下面的代码：

```
set(gcf,'DefaultLineLineWidth',2.5)
get(gcf,'default')
```

得到

```
ans =
```

```
lineLineWidth: 2.5000
```

即表示当前图形窗口中所有的默认设置只有一项，即 lineLineWidth，且值为 2.5000。

其实，将一个属性值设置为默认(即 default)，也就是将该值设置为该属性所有可能值中的第一个值。

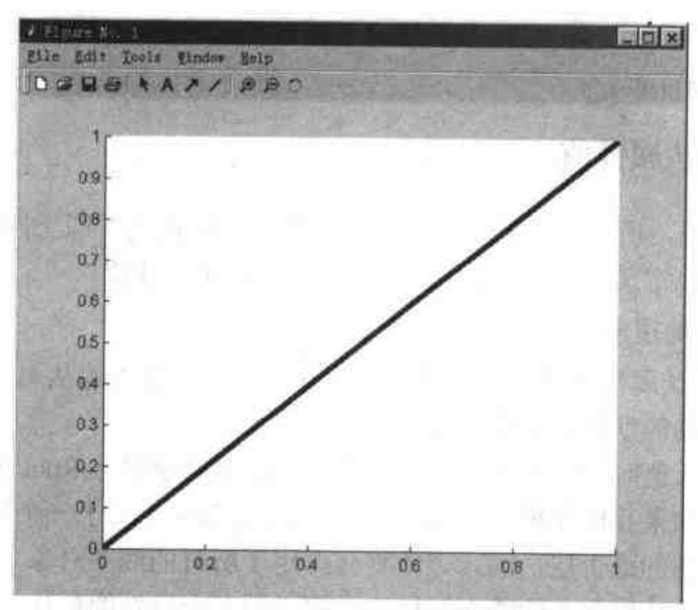


图 1.14 将线宽设置为 2.5 后所绘制的直线

1.3.3 属性值的查询

MATLAB 中，利用 get 函数可以查询对象属性的当前值。

1. 对于某个指定的属性值的查询

对于对象的某个指定的属性值的查询，其格式如下：

get(对象句柄，'属性')

例如，我们要查询当前的图形窗口对象的颜色映像表的属性，其代码如下：

```
get(gcf,'colormap')
```

回车执行后，得到如下的结果：

```
ans =
      0          0     0.5625
      0          0     0.6250
      0          0     0.6875
      0          0     0.7500
      0          0     0.8125
                 ⋮
 0.8125          0          0
 0.7500          0          0
 0.6875          0          0
```

```
0.6250    0    0
0.5625    0    0
```

2. 对象所有属性值的查询

对于对象的所有属性值的查询，其格式如下：

get(对象句柄)

例如，我们要查询当前的坐标轴对象的所有的属性，其代码如下：

```
get(gca)
```

回车执行后，得到如下的结果：

```
AmbientLightColor = [1 1 1]
Box = off
CameraPosition = [-40.6571 -43.6012 38.641]
CameraPositionMode = auto
CameraTarget = [5 4 4]
        ⋮
UserData = [ ]
Visible = on
```

另外，我们也可以把由 get 函数得到的对象的多项属性值赋给一个变量，MATLAB 自动将其创建成了一个结构体，对象的各项属性即构成了结构体中的各个元素。如果我们要访问对象的某一项属性，只需引用"结构体变量.属性名"即可。需要注意的是，原本属性名是不区分大小写的，但是该处属性名要区分大小写，否则会提示出错。

例如，执行下面的代码：

```
h=plot([1 2 3],[2 5 7]);
a=get(h);
```

表示我们将由 plot 函数创建的线条对象的属性值赋给了变量 a。再执行下面的代码：

```
a.Color
```

得到该线条对象的颜色属性值为

```
ans =
      0    0    1
```

如果执行下面的代码：

```
a.color
```

则会提示下面的错误信息：

```
??? Reference to non-existent field 'color'.
```

3. 属性的系统设定值(也称为工厂设定值)的查询

由于 MATLAB 为所有的图形对象的所有属性都指定了一个特定的值，因此当用户没有把属性值作为参数使用或没有指定其默认值时，所有的属性值都取系统指定的这个特定的值。

执行下面的代码：

```
a=get(0,'factory')
```

将得到所有属性的系统设定值,而且这些属性值都作为其中的一个元素赋给了结构体变量 a。

另外，利用下面的代码，我们也可以获得单个属性的系统设定值。

```
get(0, 'factory 对象类型属性名')
```

例如，执行下面的代码，我们将得到坐标轴对象的颜色属性的系统设定值：

```
get(0,'factoryaxescolor')
```

需要注意的是，调用函数 get 来得到属性的系统设定值的过程中，对象句柄只能为 0，即只能是根对象。

1.4 图形对象句柄的访问

MATLAB 为其所创建的所有的图形对象都赋予了一个句柄。所有的构造函数和一些高级函数的返回值即为其所创建对象的句柄。如果用户需要访问某一对象的属性(例如在.M 文件中)，则应在创建对象时就将其句柄保存在某一用户指定的变量中，以备以后调用。

当然，MATLAB 中也提供了专门的函数(即 findobj 函数)来获取现有对象的句柄；另外，罗列其父对象的 Children 属性也可以获取现有对象的句柄。

1.4.1 图形对象句柄的取值

每个图形对象句柄都有一个特定的值，但不同的对象，句柄的取值类型也略有不同。

Root 根对象的句柄取值恒为 0。

Figure 对象即图形窗口对象，其 IntergerHandle 属性控制该对象所获取的句柄的取值类型。图形窗口对象的句柄取值通常有以下两种情况：

(1) 整数。此为默认的情况，显示在窗口标题栏中，如'Figure No.1'中的 1 即为当前图形窗口对象的句柄的取值。此时 IntergerHandle 属性为'on'。

(2) 具有完全 MATLAB 内部精度的浮点数。此时 IntergerHandle 属性为'off'。

此外，所有其它的图形对象的句柄取值均为浮点类型。在引用这些句柄时，一定要维护其精度的完整。

1.4.2 句柄图形的当前性

在 MATLAB 中，句柄图形有一个非常重要的概念就是当前性(即 BeingCurrent)。例如，当前的图形窗口即为接受绘图函数输出的窗口。同样，当前坐标轴就是创建坐标子对象的命令的输出目标，而当前的图形对象即为最后创建的图形对象，或最后被鼠标点中的图形对象。

通常情况下，MATLAB 保存三个"当前句柄"，其关系如图 1.15 所示。

这些属性能够使得用户方便地获取这些关键对象的句柄，其方法如下：

(1) get(0, 'CurrentFigure')——获取当前图形窗口对象的句柄。

(2) get(gcf, 'CurrentAxes')——获取当前图形窗口对象中当前坐标轴对象的句柄。

(3) get(gcf, 'CurrentObject')——获取当前图形窗口对象中当前对象的句柄。

对于上述三种格式，MATLAB 提供了三个简化的格式，分别对应于：gcf、gca 和 gco。

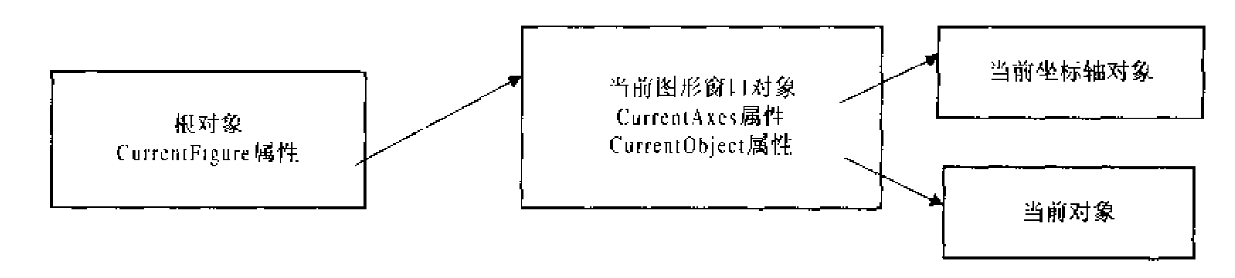


图 1.15　MATLAB 的句柄结构

我们可以利用上面的这些命令作为句柄参数传给相关的函数。例如，可以首先单击一个线条对象，然后调用 gco 命令，将其返回的句柄作为参数传递给 set 函数，代码如下：

```
set(gco, 'Marker', 'square')
```

而列出当前坐标轴对象的所有属性值的代码如下：

```
set(gca)
```

此外，用户可以获取当前坐标系下所有图形对象的句柄，代码如下：

```
h=get(gca, 'Children');
```

然后，据此确定对象类型，代码如下：

```
get(h, 'type')
```

1.4.3　通过属性值查找对象

MATLAB 的 findobj 函数提供了一种用于快速遍历对象从属关系表并获取具有特定属性值的对象句柄的方法。如果用户没有指定起始对象，那么 findobj 函数就从根对象开始查找。

findobj 函数的调用格式如下：

```
h = findobj
h = findobj (ObjectHandles)
h = findobj ('P1Name',P1Value,...)
h = findobj (ObjectHandles, 'P1Name', P1Value,...)
h = findobj (ObjectHandles, 'flat', 'P1Name', P1Value,...)
```

其中：

h = findobj 表示返回根对象和它的所有子对象的句柄，返回值为一个列向量。

h = findobj (ObjectHandles)表示返回由句柄 ObjectHandles 所确定的对象表中的对象以及它们的所有子对象的句柄。

h = findobj ('P1Name',P1Value,...)表示从根对象开始遍历对象从属关系表，从而获取具有特定属性值的对象句柄。

h = findobj (ObjectHandles, 'P1Name', P1Value,...)表示搜索的范围只能是由句柄 ObjectHandles 所确定的对象表中的对象以及它们所有的子对象。

h= findobj (ObjectHandles, 'flat', 'P1Name', P1Value,...) 表示搜索的范围只能是由句柄 ObjectHandles 所确定的对象表中的对象，而不需要搜索它们的子对象。

例 1.4.1　创建一个带有标注的正弦曲线

```
t=0:pi/20:2*pi:
```

```
plot(t,sin(t));
xlabel('t=0 to 2pi');
ylabel('sin(t)');
title('0 到 2pi 的正弦曲线');
text(pi/4,sin(pi/4),'\leftarrowsin(t)=0.707');
```

显示的结果如图 1.16 所示。

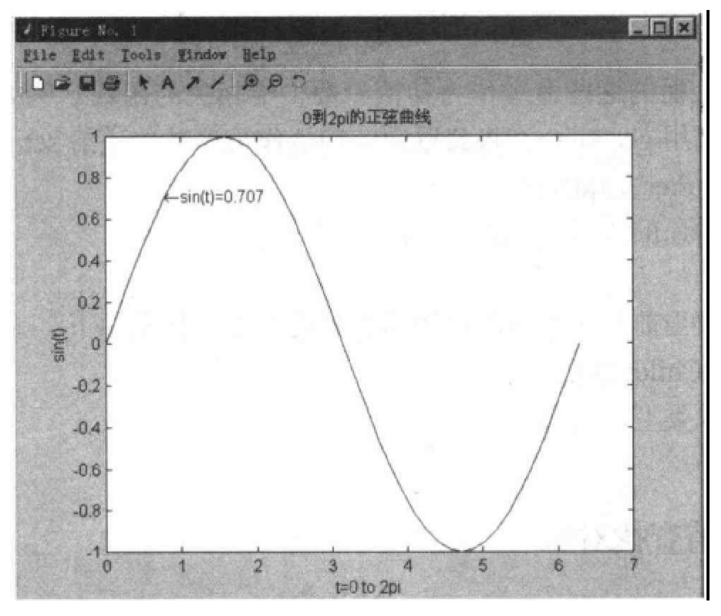


图 1.16 正弦曲线图

下面我们通过 findobj 函数来获取文本对象的句柄，进而实现对文本字符串的移动，代码如下：

```
text_handle=findobj('string','\leftarrowsin(t)=0.707');
set(text_handle,'position',[3/4*pi,sin(3/4*pi),0]);
```

文本移动后的结果如图 1.17 所示。

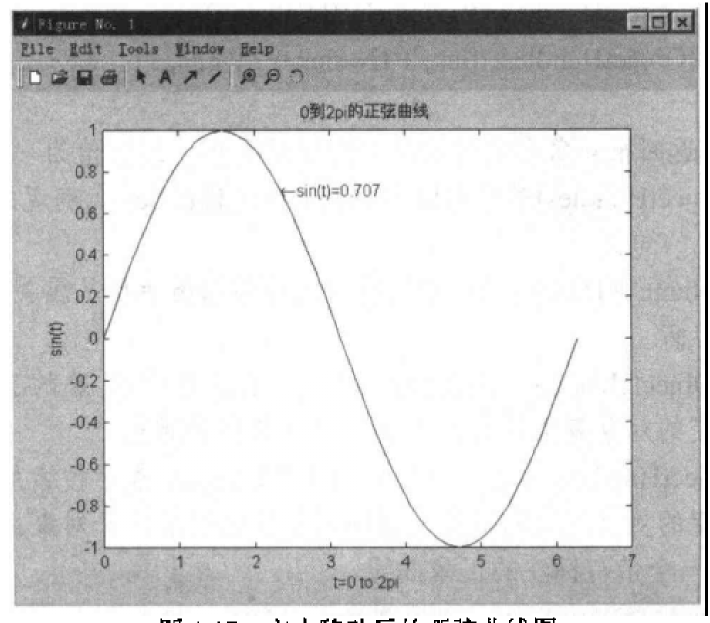


图 1.17 文本移动后的正弦曲线图

1.4.4 图形对象的拷贝

利用 MATLAB 中的 copyobj 函数，可以将一个图形对象从其父对象中拷贝到另一个对象中，从而成为该对象的子对象。当然，要实现这种拷贝，拷贝对象与新的父对象之间必须是一种正确的父对象/子对象关系，即要符合对象从属关系。

copyobj 函数的调用格式如下：

obj_handle=copyobj(h,p)

需要说明的是，h 和 p 的取值有三种情况。

(1) 当 h 和 p 都是矢量时，两者的维数必须相同，所返回的对象句柄也是一个与它们的维数相同的矢量，且 obj_handle(i)代表将 h(i)拷贝到新的父对象 p(i)后得到的新的句柄。

(2) h 是矢量，而 p 是标量时，所返回的对象句柄是与 h 维数相同的矢量，且 obj_handle(i)代表将 h(i)拷贝到新的父对象 p 后得到的新的句柄。

(3) 当 h 是标量，而 p 是矢量时，所返回的对象句柄是与 p 维数相同的矢量，且 obj_handle(i)代表将 h 拷贝到新的父对象 p(i)后得到的新的句柄。

由以上的说明可以看出，拷贝后的对象与原来的对象之间存在着两个不同：父对象不同、对象句柄不同。

另外，需要注意的是，当拷贝具有子对象的对象时，该对象所有的子对象将同时被拷贝。

例 1.4.2 把一个正弦曲线拷贝到一个指定的坐标轴对象中。

```
figure;
t=0:pi/20:2*pi;
plot(t,sin(t));


f=axes;
copyobj(h,f);
```

1.4.5 图形对象的删除

利用 MATLAB 中的 delete 函数，可以删除图形对象。

delete 函数的调用格式如下：

delete(obj_handle)

由上面的格式可以看出，要想删除一个图形对象，必须先获得该对象的句柄。

另外，需要说明的是，当删除一个具有子对象的对象时，该对象所有的子对象将同时被删除。

例 1.4.3 把当前的坐标轴连同它的子对象——一条正弦曲线同时从图形窗口中删除。

```
t=0:pi/20:2*pi;
plot(t,sin(t));
delete(gca);
```

图形编程基础篇

第二章　MATLAB 的图形绘制

MATLAB 为用户提供了大量的绘图函数，使用户可以方便地实现数据的可视化，如在直角坐标系中或极坐标系中绘制直线、条形图、柱状图、轮廓线和表面网格图等等。本章将对这些绘图函数的用法进行较为详细的介绍，使读者能够对 MATLAB 提供的这些高级绘图函数有一个较为深刻的认识，从而绘制出自己需要的图形。

本章主要内容：

★ 基本曲线的绘制
★ 条形图的绘制
★ 柱状图的绘制
★ 区域图的绘制
★ 饼图的绘制
★ 离散数据的图形绘制
★ 方向和速度矢量图的绘制
★ 轮廓图的绘制
★ 动画的绘制

2.1　基本曲线的绘制

MATLAB 提供了大量的图形绘制函数，用于将矢量数据以曲线图形的方式进行显示，而且用户还可以通过设置对象的属性(如线条类型、线条的颜色等等)来控制显示的外观。

表 2.1 列出了 MATLAB 中绘制基本曲线图形的函数。

表 2.1　基本的绘制曲线的函数

绘图函数	用　途
plot	在线性坐标系中绘制二维数据
plot3	在线性坐标系中绘制三维数据
loglog	在对数坐标系中绘制图形
semilogx	在半对数坐标系中绘制图形(x 轴为对数比例)
semilogy	在半对数坐标系中绘制图形(y 轴为对数比例)
plotyy	在双 y 轴坐标系中绘制图形

下面简单介绍这些函数的用法。

2.1.1 plot 函数

plot 函数的调用格式如下：

```
plot(Y)
plot(X1,Y1,...)
plot(X1,Y1,LineSpec,...)
plot(...,'PropertyName',PropertyValue,...)
h = plot(...)
```

其中：

在 plot(Y)中，如果 Y 为一个矢量，则绘制连接(i ,Y(i))的折线；若 Y 中的各元素为复数，则绘制连接(real(Y(i)),imag(Y(i)))的折线；若 Y 为矩阵，则将每一列都视为一个矢量绘制一条折线。

在 plot(x,y)中，若 x 和 y 都是向量，则绘制出连接(x(i)，y(i))的折线。x 和 y 也可以是矩阵，但要注意维数的匹配。

参数 LineSpec 表示定义线形、标记符号、颜色。以后凡遇到该参数，若无特别说明，都是此含义。

2.1.2 plot3 函数

plot3 函数的调用格式如下：

```
plot3(X1,Y1,Z1,...)
plot3(X1,Y1,Z1,LineSpec,...)
plot3(...,'PropertyName',PropertyValue,...)
h = plot3(...)
```

其中，X1，Y1 和 Z1 若是三个长度相同的矢量，则绘制连接(X(i),Y(i),Z(i))的折线；若为大小相同的矩阵，则每一列都绘制一条折线。

2.1.3 loglog 函数、semilogx 函数和 semilogy 函数

这三个函数的调用格式与 plot 函数的格式相同，只不过是显示的坐标轴的比例不同。

2.1.4 plotyy 函数

plotyy 函数的调用格式如下：

```
plotyy(X1,Y1,X2,Y2)
plotyy(X1,Y1,X2,Y2,'function')
plotyy(X1,Y1,X2,Y2,'function1','function2')
[AX,H1,H2] = plotyy(...)
```

其中：

参数'function'代表绘制数据的方式，可选择'plot'，'semilogx'，'semilogy'，'loglog'，'stem'等等。

[AX,H1,H2] = plotyy(...)中的 AX 是返回的两个坐标轴的句柄，H1 和 H2 为两个图形对象的句柄。

例 2.1.1　绘制一个带标注的双 y 轴图形。

```
t=0:1000;
A=1000;
a=2.005;
b=2.005;
z1=A*exp(.a*t);
z2=cos(b*t);
[haxes,hline1,hline2]=plotyy(t,z1,t,z2,'semilogy','plot');  %获取两个坐标轴和两个线条对象的句柄
axes(haxes(1));
ylabel('semilog plot');        %标注左边的 y 轴
axes(haxes(2));
ylabel('linear plot');         %标注右边的 y 轴
```

显示的结果如图 2.1 所示。

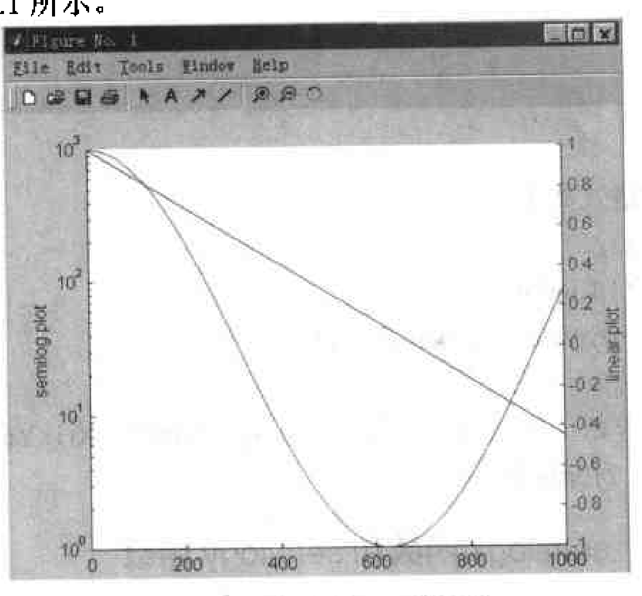


图 2.1　带标注的双 y 轴图形

2.2　条形图的绘制

2.2.1　二维垂直的条形图

在 MATLAB 中，用函数 bar 来绘制二维条形图。该函数的调用格式如下：

```
bar(Y)
bar(x,Y)
bar(...,width,'style',LineSpec)
```

其中:

bar(Y)中的 Y 为 m×n 的矩阵，绘制后的条形图中有 m 组，每组有 n 个条形，且第 i 组在 x 轴的中心坐标为 i，条形的高度为所对应的元素的大小。

bar(x,Y)与 bar(Y)的意义类似，但第 i 组在 x 轴的中心坐标为 x(i)。另外，需要注意的是，此处 x 中的各元素必须是降幂或升幂排列。

bar(...,width,'*style*',LineSpec)中的参数 width 代表条形的宽度，当 width 的值大于 1 时，条形将会出现交叠，默认值为 2.8。参数'*style*'用来定义条形的类型，可选值为'group'或'stack'，默认值为'group'，当选'stack'时，对于 m×n 的矩阵只绘制 n 组条形，每组一个条形，且该条形的高度为这一列中所有元素的和。参数 LineSpec 用来定义条形的颜色。

例 2.2.1　绘制一个常规的二维条形图。

```
x=[0.5   1.5   2];
y=[2   4   1;
   3   6   2;
   2   4   8];
bar(x,y,1.2);
```

显示的结果如图 2.2 所示。

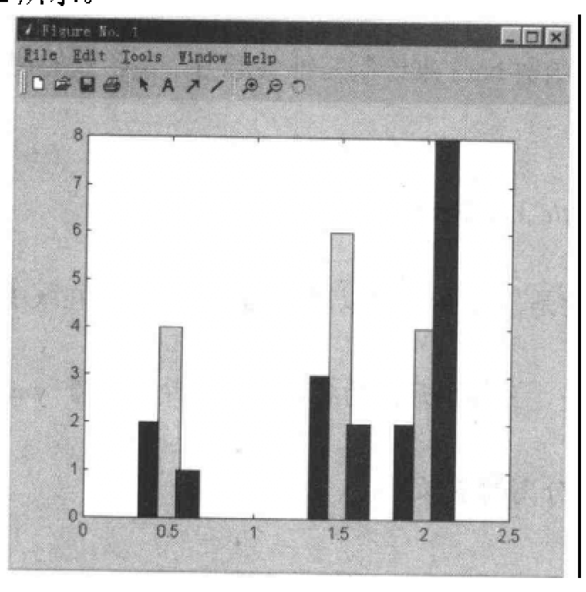


图 2.2　常规的二维条形图

例 2.2.2　绘制一个堆叠的二维条形图(此时将参数'style'设置为'stack')。

```
x=[0.5   1.5   2];
y=[2   4   1;
   3   6   2;
   2   4   8];
bar(x,y,1.2,'stack');
```

显示的结果如图 2.3 所示。

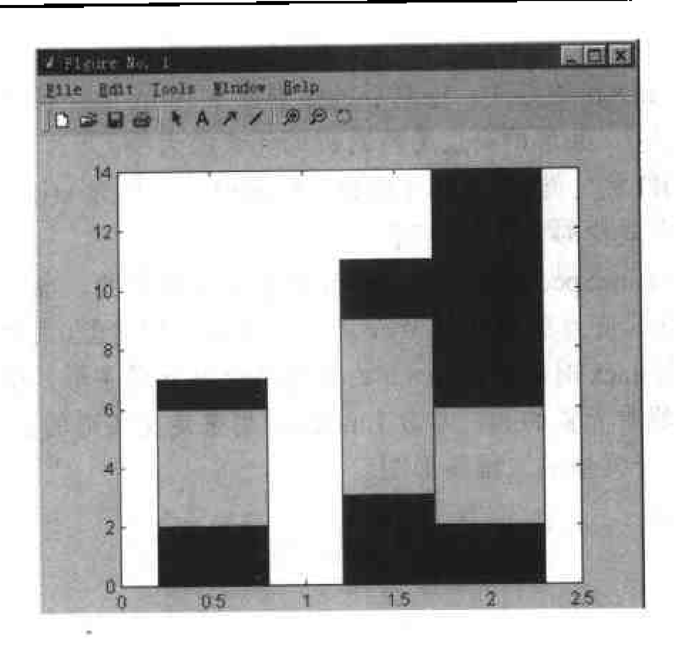


图 2.3　堆叠的条形图

2.2.2　三维垂直的条形图

在 MATLAB 中，用函数 bar3 来绘制三维条形图。该函数的调用格式如下：

```
bar3(Y)
bar3(x,Y)
bar3(...,width,'style',LineSpec)
```

其中：

与二维条形图不同的是，'*style*'还可以取'detached'，此时在 x 轴方向各个实心块是彼此分离的。

另外，需要说明的是，三维条形图的各组的实心块是沿着 y 轴分布的，而不同的组是沿着 x 轴排列的。

例 2.2.3　创建一个分离的条形图。

```
x=[0.5  1.5  2];
y=[2  4  1;
   3  6  2;
   2  4  8];
bar3(x,y,'detached');
xlabel 'x 轴';
ylabel 'y 轴';
zlabel 'z 轴';
set(gca,'xtick',[1 2 3]);     %设置 x 轴的标度
```

显示的结果如图 2.4 所示。

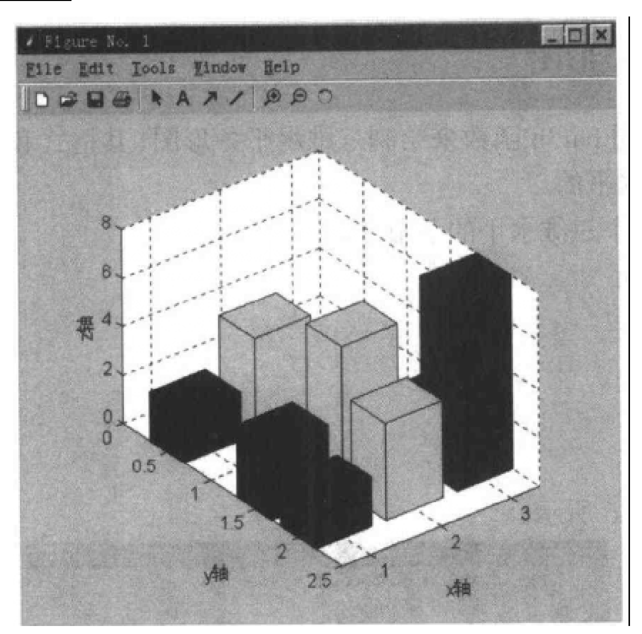


图 2.4 分离的三维条形图

2.2.3 二维水平的条形图

MATLAB 中，用 barh 函数来绘制二维水平条形图，其格式和用法与函数 bar 相同，只是绘出的条形图是水平的。

例 2.2.4 绘制一个二维水平且堆叠的条形图。

```
x=[0.5  1.5  2];
y=[2  4  1;
   3  6  2;
   2  4  8];
barh(x,y,'stacked');
```

显示的结果如图 2.5 所示。

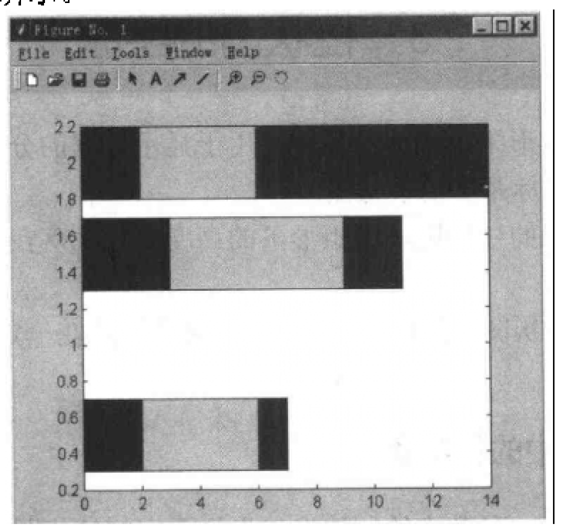


图 2.5 水平且堆叠的条形图

2.2.4 三维水平的条形图

在 MATLAB 中，用 bar3h 函数来绘制三维水平条形图，其格式和用法与函数 bar3 相同，只是绘出的条形图是水平的。

例 2.2.5 绘制一个三维水平的条形图。

```
x=[0.5  1.5  2];
y=[2  4  1;
   3  6  2;
   2  4  8];
bar3h(x,y,'group');
```

显示的结果如图 2.6 所示。

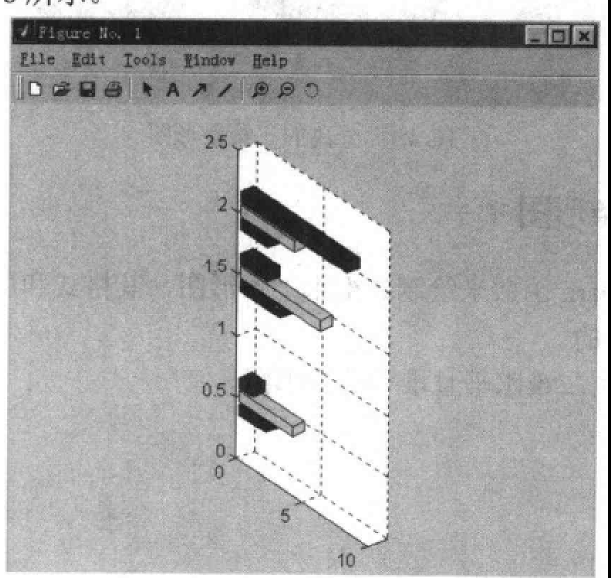


图 2.6 三维水平条形图

2.3 柱状图的绘制

柱状图是由一系列矩形条组成的，其主要用于显示数据的分布情况。柱状图有迪卡尔坐标系中的柱状图和极坐标系中的柱状图。

对于迪卡尔坐标系中的柱状图，用矩形条的高度来代表落在一定取值范围内的数据的个数。

对于极坐标系中的柱状图，则用小扇形的半径来表示落在一定的角度(单位是弧度)范围内的角度数据的个数。

2.3.1 迪卡尔坐标系中的柱状图

在 MATLAB 中，用 hist 函数来绘制迪卡尔坐标系中的柱状图。该函数的调用格式如下：

n = hist(Y)

n = hist(Y,x)

n = hist(Y,nbins)

其中：

n = hist(Y)中的 Y 既可以是向量，也可以是矩阵。当 Y 为向量时，该函数自动将 x 轴分为 10 个等间隔区域，然后在这 10 个区域上绘制矩形条，矩形条的高度代表落在这一区域的元素的个数；当 Y 为矩阵时，则每一列都会相应地绘制一组矩形条。

n = hist(Y,x)中的 x 为 n 维向量，则将创建 n 个区域，每个区域的中心在 x 的各个元素所在的位置上。

n = hist(Y,nbins)中的 nbins 指定了区域的个数。

所有的 n 都是用来存储落在各个区域的元素的个数。

例 2.3.1　绘制一个迪卡尔坐标系中的柱状图。

```
Y=randn(10000,3);        %随机生成一个 10000×3 的矩阵
hist(Y,5);
```

显示的结果如图 2.7 所示：

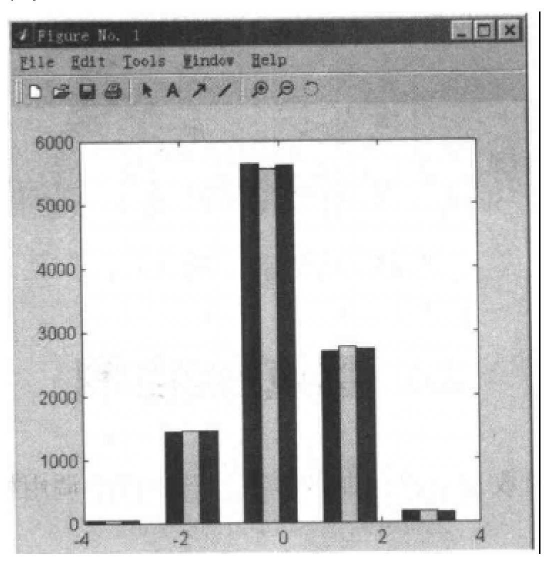


图 2.7　迪卡尔坐标系中的柱状图

2.3.2　极坐标系中的柱状图

在 MATLAB 中，用 rose 函数来绘制极坐标系中的柱状图。该函数的调用格式如下：

rose(theta)

rose(theta,x)

rose(theta,nbins)

其中：

参数 theta 为角度数据；

x 定义了小扇形的位置，它的维数决定了小扇形的个数；

nbins 定义了小扇形的个数，默认值为 20。

例 2.3.2　绘制一个极坐标系中的柱状图。

```
theta=[1.2 1.1 3.4 2.1 1.5 1 6 2.6 2.4 1 8];
```

```
x=2.5:2.3:3.2;
rose(theta,x);
hline=findobj(gca,'type','line');        %获取当前坐标系中的线条对象的句柄
set(hline,'linewidth',2.0);              %将线条对象的线宽属性值设为 2.0
```

显示的结果如图 2.8 所示。

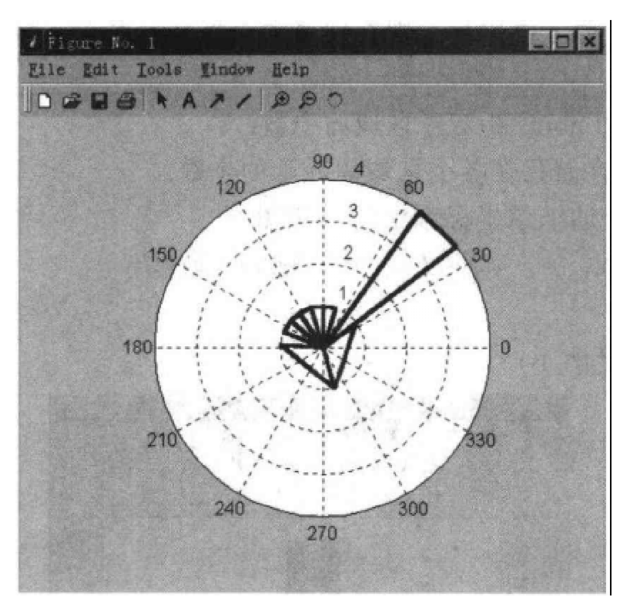


图 2.8 极坐标系中的柱状图

2.4 区域图的绘制

在 MATLAB 中，用函数 area 来绘制区域图。该函数的调用格式如下：

```
area(Y)
area(X,Y)
area(...,ymin)
area(...,'PropertyName',PropertyValue,...)
h = area(...)
```

其中：

area(Y)中的 Y 既可以是向量，也可以是矩阵。当 Y 为向量时，该函数先绘制一条连接(Y(i),i)的折线，然后对该曲线与 x 轴之间的区域进行填充；当 Y 为矩阵时，则针对该矩阵的每一列都绘制一条曲线，然后对曲线与 x 轴和曲线与曲线之间的区域进行填充。需要注意的是，对应第 i 列的折线上的点坐标为((Y(j,i)+Y(j,i-1))，j)，即要加上前一列所对应的元素。

area(X,Y)所表示的意义与 area(Y)类似，只不过折线上的点坐标的 x 坐标由 X 来确定。

area(...,ymin)中的 ymin 表示填充的区域为所绘制折线与 y=ymin(默认情况下为 y=0，即 x 轴)之间的区域。

area(...,'*PropertyName*',PropertyValue,...)表示用户可以设置特定的对象属性，来控制区域

图显示的外观。

例 2.4.1 创建区域对象。

```
Y=[3  1  2
   8  7  5
   5  3  4
   9  3  2];
area(Y)
grid on
set(gca,'layer','top');
```

显示的结果如图 2.9 所示。

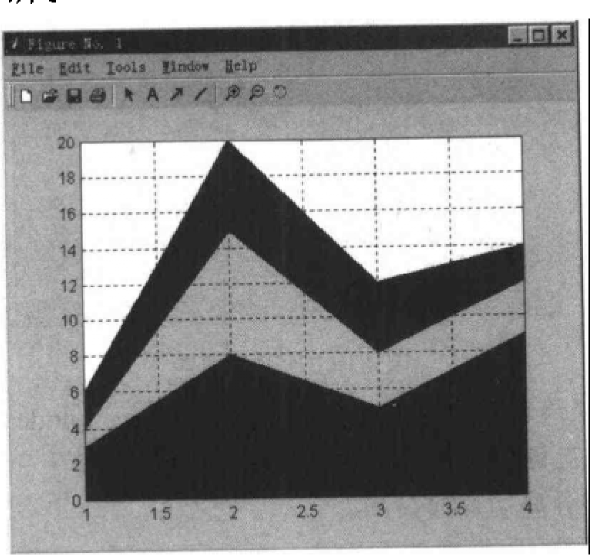


图 2.9 由矩阵数据绘制的区域图

2.5 饼图的绘制

在 MATLAB 中，饼图用于显示矢量或矩阵中的每个元素在其所有的总和中所占的百分比。

MATLAB 中绘制饼图的函数有两个，即 pie 函数和 pie3 函数，它们分别用于创建二维和三维饼图。下面分别介绍这两个函数的用法。

2.5.1 二维饼图的绘制

在 MATLAB 中，用 pie 函数来实现二维饼图的绘制。该函数的调用格式如下：

```
pie(x)
h=pie(x,explode)
h=pie(x,labels)
```

其中：

pie(x)中，x 既可以是矩阵，也可以是向量。且如果 x 中的各个元素的和大于或等于 1，

则饼图中的每个切片占整个饼图的百分比为 x(i,j)/sum；如果 x 中的各个元素的和小于 1，则绘制的是一个不完全饼图，且每个切片占整个饼图的百分比为 x(i,j)/1=x(i,j)。

例 2.5.1　绘制一个不完全饼图。

```
x=[2.1,2.3,2.4,2.1];
pie(x);
```

显示结果如图 2.10 所示。

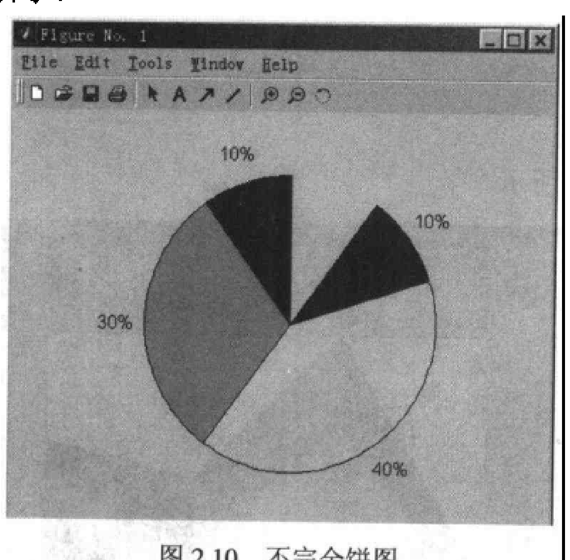


图 2.10　不完全饼图

h=pie(x,explode)用来绘制具有分离切片的饼图。参数 explode 必须与 x 的维数相同，explode 中非零元素所对应的切片即为应分离的切片。

例 2.5.2　绘制具有分离切片的饼图。

```
x=[1,2,3,4];
explode=[0,0;1,1];
pie(x,xplode);
```

显示的结果如图 2.11 所示。

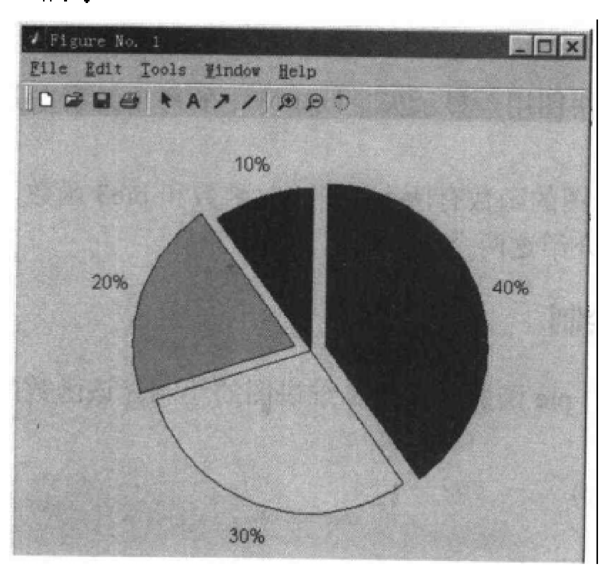


图 2.11　带分离切片的饼图

h=pie(x,labels)用来绘制具有特定标注的饼图。参数 labels 必须与 x 的维数相同，且它里面包含的元素只能是字符串。

例 2.5.3 绘制出具有特定标注的饼图。

```
x=[80 81 70 110];
labels={'一队' '二队' '三队' '四队'};
pie(x,labels);
```

显示的结果如图 2.12 所示。

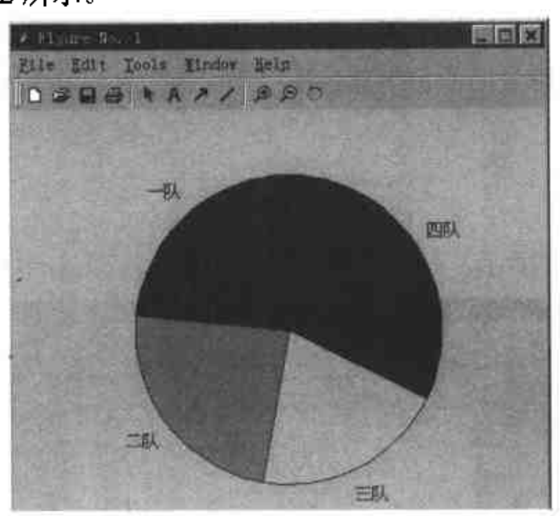


图 2.12 带有特定标注的饼图

需要注意的是，MATLAB 中饼图的标注是作为文本对象来处理的，所以如果要修改标注文本字符串或它的位置，则首先要获取它的句柄，然后修改它的属性来实现上述的目的。

2.5.2 三维饼图的绘制

在 MATLAB 中，用 pie3 函数来实现三维饼图的绘制。该函数的调用格式如下：

```
pie3(x)
h=pie3(x,explode)
h=pie3(x,labels)
```

其中，各项参数的意义与 pie 函数中各项参数的意义完全相同，此处不再赘述。

2.6 离散数据的图形绘制

工程应用和科学计算中经常会得到许多离散数据，MATLAB 提供了一些特殊的函数来将这些离散数据以图形的形式显示出来，使用户能够更加直观地看出这些离散数据的内在特点和规律。

2.6.1 二维枝干图

枝干图即将每个离散数据显示为末端带有标记符号的线条(即枝干)。在二维枝干图中，枝干线条的起点在 X 坐标轴上。

在 MATLAB 中，用 stem 函数来绘制二维枝干图。其调用格式如下：

```
stem(x,y, 'linestyle or marker or color', 'fill')
```

其中：

参数 x 和 y 表示要绘制的离散数据；

参数'linestyle or marker or color'表示用户可以自定义枝干的线形、标记符号的类型和颜色；

参数'fill'表示对于有些标记符号，我们还可以对其进行填充。

例 2.6.1 绘制出一个二维枝干图，且其线形被指定为虚线(用':'表示)，标记符号为菱形(用'd'表示)，线条和标记符号的颜色为蓝色(用'b'表示)。

```
x=0:0.2:2;
y=sin(x);
stem(x,y,':db','fill');
```

显示的结果如图 2.13 所示。

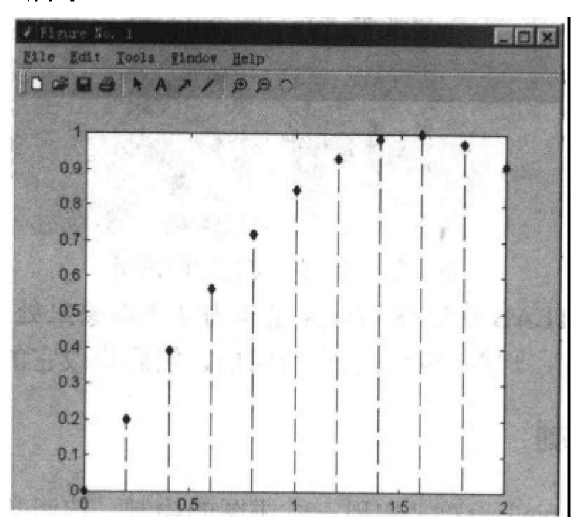


图 2.13 二维枝干图

2.6.2 三维枝干图

在 MATLAB 中，用 stem3 函数绘制起点在 xy 平面上的三维枝干图。其调用格式如下：

```
stem3(z)
stem3(x,y,z, 'linestyle or marker or color', 'fill')
```

其中：

stem3(z)表示将只在 x=1(当 z 为一个列向量时)或 y=1(当 z 为一个行向量时)处绘制一行枝干图。

第二种格式中的各项参数与二维枝干图的解释类似。

例 2.6.2 利用三维枝干图显示快速傅立叶变换的计算过程。

```
th=(0:127)/128*2*pi;
x=cos(th);
y=sin(th);                    %计算复平面上的单位圆
```

```
f=(abs(fft(ones(10,1),128)))';          %计算一步频率响应的幅值
stem3(x,y,f,'*');                       %绘制三维枝干图
xlabel('实部');
ylabel('虚部');
zlabel('幅值');
title('频率响应');                       %进行标注
```

显示的结果如图 2.14 所示。

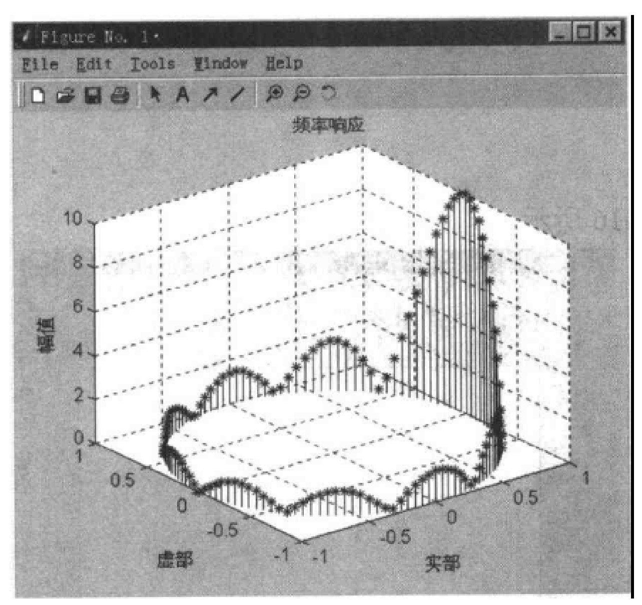


图 2.14 快速傅立叶变换的三维枝干图

如果需要换个角度查看三维枝干图，则首先执行下面的代码：

```
rotate3d on
```

然后就可以用鼠标拖动该三维枝干图，转到所希望的角度。图 2.15 给出了旋转后的该三维枝干图。

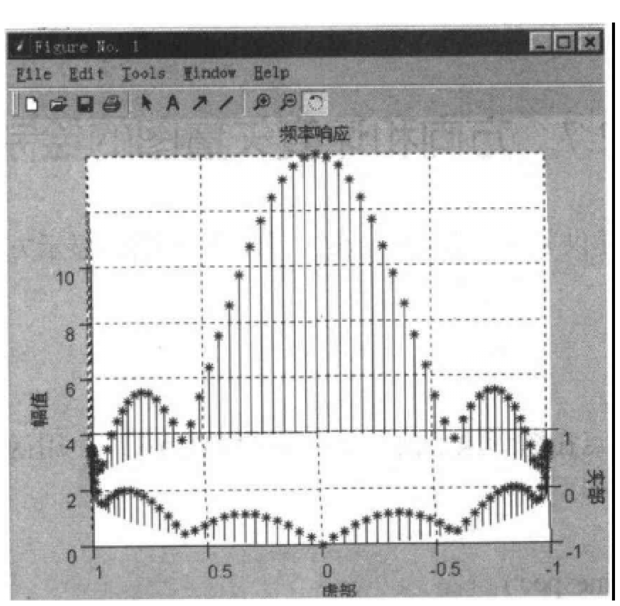


图 2.15 旋转后的三维枝干图

2.6.3 阶梯图

阶梯图以一个恒定间隔的上升沿显示数据。在阶梯图中，在 x 坐标轴的 x(i)到 x(i+1)之间的所有位置的 y 值都为 y(i)。

在 MATLAB 中，用函数 stairs 来绘制阶梯图。其调用格式如下：

```
stairs(x,y, 'linestyle or marker or color')
```

其中，参数'linestyle'用于指定绘制阶梯的线形、标记符号以及颜色。

例 2.6.3 绘制一个用户指定线形和标记符号以及颜色的阶梯图。

```
x=1:10;
y=sin(x);
stairs(x,y,':rd');
```

显示的结果如图 2.16 所示。

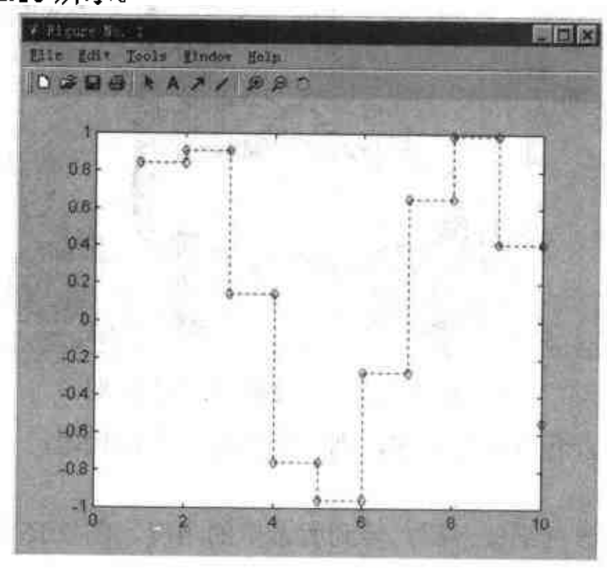


图 2.16 阶梯图

2.7 方向和速度矢量图的绘制

在 MATLAB 中，提供了一些特殊的函数，专门用于绘制显示方向矢量或速度矢量数据的图形。

2.7.1 罗盘图

罗盘图用来显示起点在图形原点的矢量。在 MATLAB 中，用函数 compass 来绘制罗盘图，该函数的调用格式如下：

```
compass(x)
compass(x,y, 'linespec')
```

其中：

compass(x,y, 'linespec')表示绘制以图形原点为起点，以(x(i),y(i))为终点的矢量箭头图。最后一个参数表示用户可以自定义箭头矢量的绘制特征(如线形、颜色等等)。

compass(x)则等价于 compass(real(x),iamg(x))。

需要注意的是，该函数采用的是迪卡尔坐标系，而且是在圆形栅格上绘制图形的。但圆形栅格的最外层标注的却是在极坐标系下的角度，各个小圆上标注的则是矢径的长度。所以当给定极坐标时，要先将其转换为直角坐标(用 pol2cart 函数即可)，再绘制罗盘图。

例 2.7.1 绘制风向与风力的罗盘图。

```
w_direction=[45 90 90 45 180];
w_intensity=[6 7 3 8 5];                                %以上为代表风向和风力的矢量
arcw_direction=w_direction*pi/180;
[x,y]=pol2cart(arcw_direction,w_intensity);             %将极坐标转换为直角坐标
compass(x,y);                                           %绘制罗盘图
title('长沙地区五小时内的风向和风力图');                 %标注
```

显示的结果如图 2.17 所示。

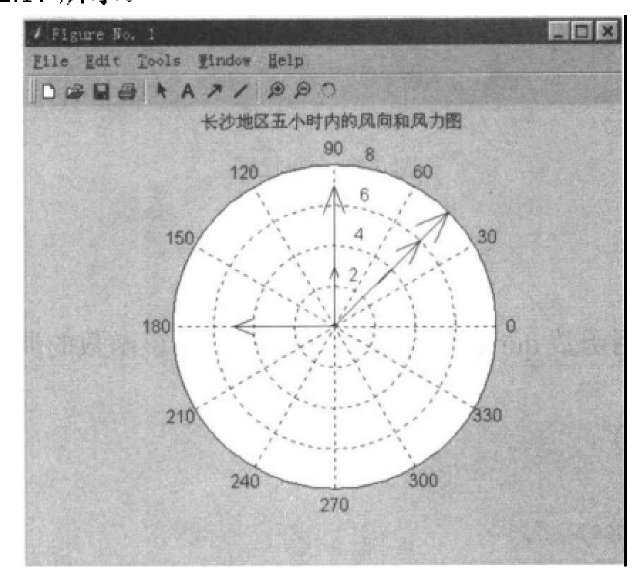


图 2.17 风向和风力的罗盘图

2.7.2 羽状图

羽状图也是以箭头的形式显示一系列矢量，与罗盘图不同的是，这些矢量的起点不是在原点，而是均匀地分布在一条与 x 轴平行的直线上。

在 MATLAB 中，用函数 feather 来绘制羽状图。该函数的调用格式如下：

```
feather(x)
feather(x,y, 'linespec')
```

关于这两种格式的说明与在罗盘图的类似，此处不再赘述。

例 2.7.2 绘制羽状图。

```
x=[3 4 5 6];
```

```
y=[2 3 4 5];
feather(x,y);
```

显示的结果如图 2.18 所示。

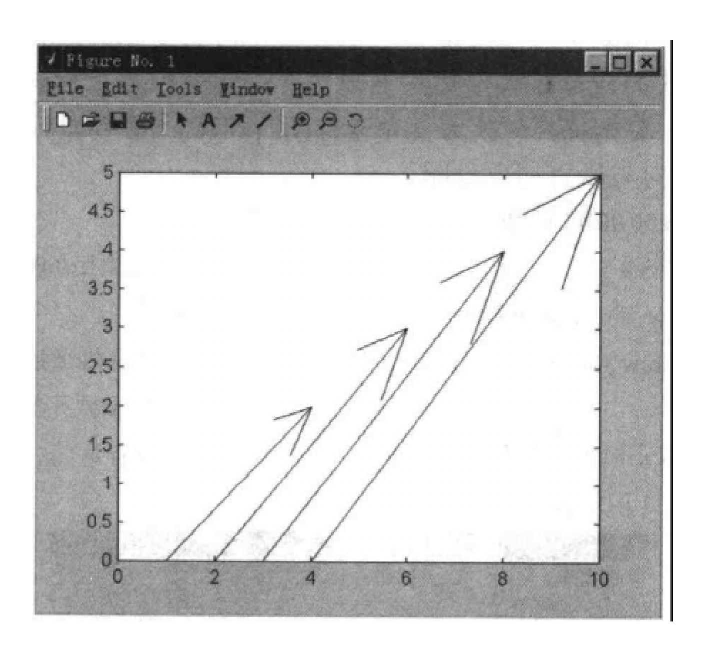


图 2.18　羽状图

2.7.3　二维箭头图

在 MATLAB 中，用函数 quiver 来绘制二维箭头图。该函数的调用格式如下：

```
quiver(U,V)
quiver(X,Y,U,V)
quiver(...,scale)
quiver(...,LineSpec)
quiver(...,LineSpec,'filled')
h = quiver(...)
```

其中：

X 和 Y 中的元素组成的点(x,y)作为起点，U 和 V 中的元素组成的点(u,v)作为终点，且其数值是相对起点而言的。

scale 代表缩放比例，为 0 时，矢量是原长度；默认值为 1，此时自动调整缩放比例，以防止交叠；为 2.5 时，长度为为 1 时的一半。

例 2.7.3　绘制一个二维箭头图。

```
x=[1 2 3];
y=[1 2 3];
u=[7 8 9];
v=[4 5 6];
quiver(x,y,u,v,0);
```

显示的结果如图 2.19 所示。

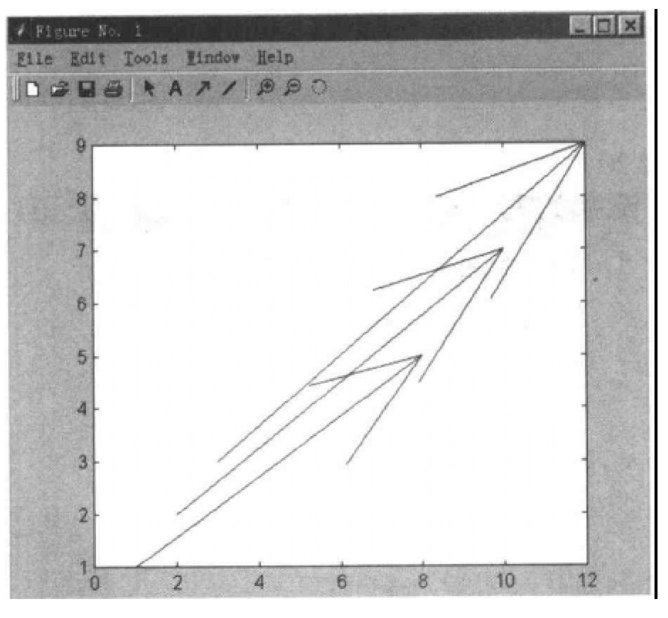


图 2.19 二维箭头图

2.7.4 二维箭头图

MATLAB 中，用函数 quiver3 来绘制三维箭头图。该函数的调用格式如下：

```
quiver3(Z,U,V,W)
quiver3(X,Y,Z,U,V,W)
quiver3(...,scale)
quiver3(...,LineSpec)
quiver3(...,LineSpec,'filled')
h = quiver3(...)
```

其中，各项参数的说明与二维箭头图的类似，只是在三维坐标系中多了一个 z 坐标。

例 2.7.4 用三维箭头图显示抛物线。

```
vz=10;
a=.32;
t=0:2.1:1;
z=vz*t+1/2*a*t.^2;
vx=2;
x=vx*t;
vy=3;
y=vy*t;
u=gradient(x);
v=gradient(y);
w=gradient(z);
scale=0;
```

```
quiver3(x,y,z,u,v,w,scale)
xlabel('x 轴');
ylabel('y 轴');
zlabel('z 轴');
```

显示的结果如图 2.20 所示。

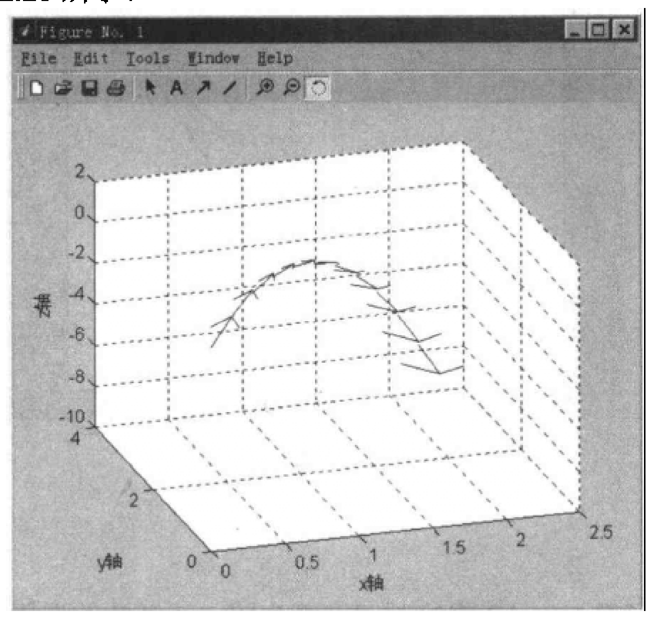


图 2.20 三维箭头图

2.8 轮廓图的绘制

MATLAB 中的轮廓图即指将相对于某一平面(即 xy 平面)具有同一高度的点连成一条曲线，而该高度是通过高度矩阵来反映的。在 MATLAB 中有两类轮廓图，即二维轮廓图和三维轮廓图。

2.8.1 二维轮廓图

在 MATLAB 中，用函数 contour 来绘制二维轮廓图。该函数的调用格式如下：

```
contour(Z)
contour(Z,n)
contour(Z,v)
contour(X,Y,Z)
contour(X,Y,Z,n)
contour(X,Y,Z,v)
contour(...,LineSpec)
[C,h] = contour(...)
```

其中：

Z 为高度矩阵，X 和 Y 代表 x 轴和 y 轴的取值，且 X 和 Y 的维数与 Z 的维数必须匹配，

若 X 和 Y 是矩阵，则必与 Z 的维数相同，此时将绘制一个表面对象；

参数 n 代表轮廓线的条数；

参数 v 为一个矢量，用来定义在何处绘制轮廓线，各个元素的取值即代表了轮廓线的基准值；

[C,h] = contour(...)表示返回一个轮廓矩阵 C 和一个图形句柄矢量 h。利用这两个参数，可调用函数 clabel 对轮廓图进行标注。

例 2.8.1 绘制一个具有标注的二维轮廓图。

```
[X,Y,Z]=peaks;
[C,h]=contour(X,Y,Z,15,'r*');
clabel(C,h);
title('具有 15 条轮廓线的 peaks 函数');
```

显示的结果如图 2.21 所示。

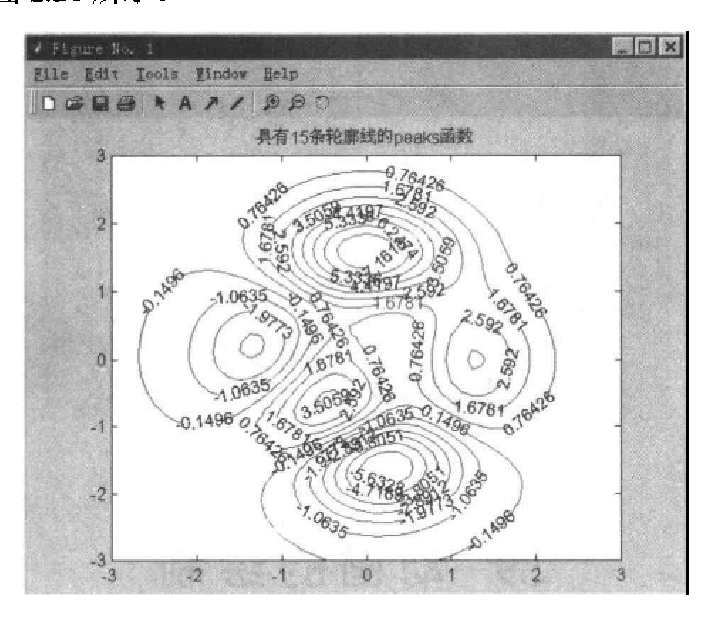


图 2.21 带标注的二维轮廓图

2.8.2 三维轮廓图

在 MATLAB 中，用函数 contour3 来绘制三维轮廓图。该函数的调用格式如下：

```
contour3(Z)
contour3(Z,n)
contour3(Z,v)
contour3(X,Y,Z)
contour3(X,Y,Z,n)
contour3(X,Y,Z,v)
contour3(...,LineSpec)
[C,h] = contour3(...)
```

其中，关于各项参数的说明与 contour 函数相同。

例 2.8.2 绘制一个具有标注的三维轮廓图。

```
[X,Y,Z]=peaks;
[C,h]=contour3(X,Y,Z,15,'r*');
title('具有 15 条轮廓线的 peaks 函数');
xlabel 'x';
ylabel 'y';
zlabel 'z';
```

显示的结果如图 2.22 所示。

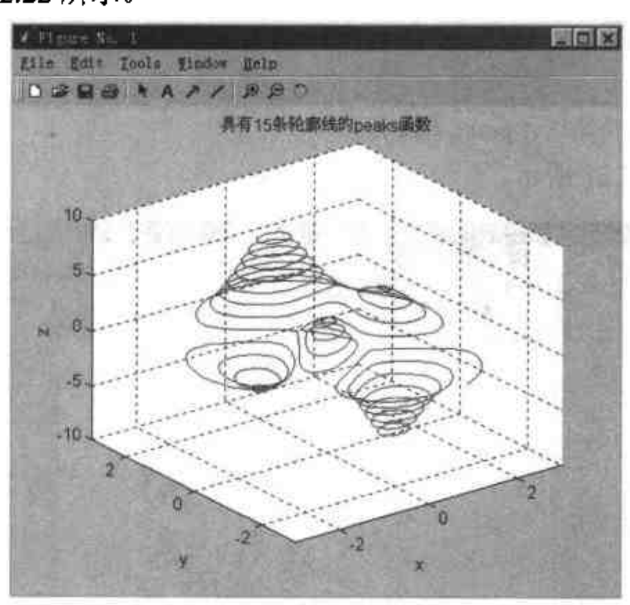


图 2.22 三维轮廓图

2.9 动画的绘制

MATLAB 中，用两种方法来创建动画。

(1) 首先保存一系列图形，然后按照一定的顺序像电影一样播放。这种方式的动画称之为电影动画。

(2) 在图形窗口中按照一定的算法连续擦除和重绘图形对象。这种方式的动画称之为程序动画。

2.9.1 电影动画

MATLAB 中，创建电影动画的过程分为以下三步：

(1) 调用 moviein 函数对内存进行初始化，创建一个足够大的矩阵，使之能够容纳基于当前坐标轴大小的一系列指定的图形(此处称为帧)。

(2) 调用 getframe 函数生成每个帧。该函数返回一个列矢量，利用这个矢量，就可以创建一个电影动画矩阵。

(3) 调用 movie 函数按照指定的速度和次数运行该电影动画。

例 2.9.1 创建一个电影动画，用来演示快速傅立叶变换的过程。

```
axis equal;
M=moviein(16,gcf);
set(gca,'nextplot','replacechildren');
h=uicontrol('style','slider','position',...
    [100 10 500 20],'min',1,'max',16);
for j=1:16
    plot(fft(eye(j+16)))
    set(h,'value',j)
    M(:,j)=getframe(gcf);
end
clf
axes('position',[0 0 1 1]);
movie(M,30);
```

动画播放过程中的一个画面如图 2.23 所示。

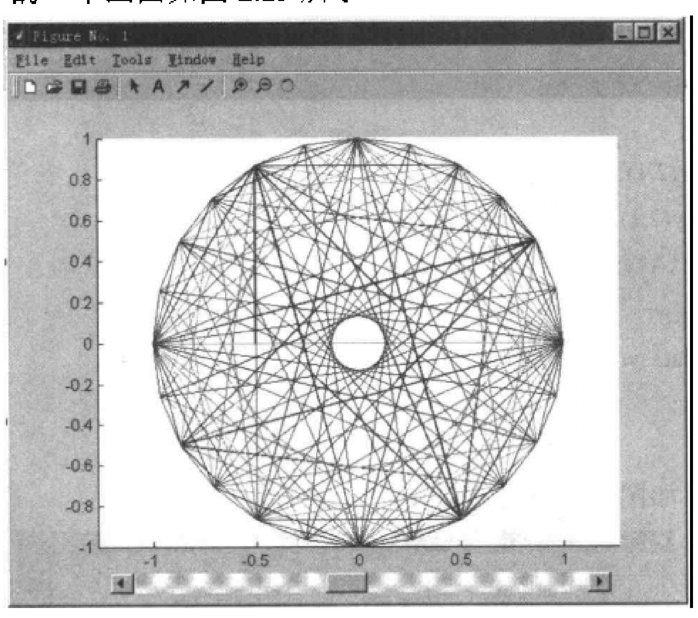


图 2.23 电影动画的播放画面

2.9.2 程序动画

在 MATLAB 中，采用重绘图形的方法来创建程序动画。改变对象的方法可以触发 MATLAB 对该对象进行重绘。利用图形对象的这种属性，创建程序动画的典型步骤有两步：

(1) 绘制一个图形对象；

(2) 在程序循环中改变该对象的 x、y、z 坐标值来实现对象的移动，从而形成动画。

另外，在创建 MATLAB 的程序动画时，图形的擦除也是很重要的。通过设置不同的擦除模式，用户可以得到各种动画效果。MATLAB 为用户提供了以下三种擦除模式：

- None——在移动图形对象时，不进行擦除。

● Background——在图形对象移走后，在原来的位置用背景色进行重绘。在这种模式下，MATLAB 将原来的对象完全擦除，包括该对象下面的所有图形。

● Xor——与第二种模式相比，这种模式只擦除对象本身。大多数的 MATLAB 程序动画都采用这种擦除模式。

例 2.9.2 创建一个程序动画，在三维空间演示一个由著名的劳仑兹非线性奇异方程所描述的无序运动。劳仑兹方程形式如下：

$$\frac{dy}{dt} = Ay$$

方程中包含一个由函数 y(t)确定的矢量和一个矩阵 A，且 A 依赖于 y，存在着下面的关系：

A=[.8/3 ,0, y(2);0,.10,10;.y(2),28,.1]

该程序的代码如下：

```
A=[.8/3 0 .10;0 .10 10;10 28 .1];
y=[35;.10;.7];
h=2.01;
p=plot3(y(1,1),y(2,1),y(3,1),'*','erasemode','none');
axis([0 50 .25 25 .25 25]);
hold on;
for i=1:8000
    A(1,3)=y(2,1);
    A(3,1)=.y(2,1);
    dita_y=h*A*y;
    y=y+dita_y;
    set(p,'xdata',y(1,1),'ydata',y(2,1),'zdata',y(3,1));
    drawnow;
end
```

程序运行的结果如图 2.24 所示。

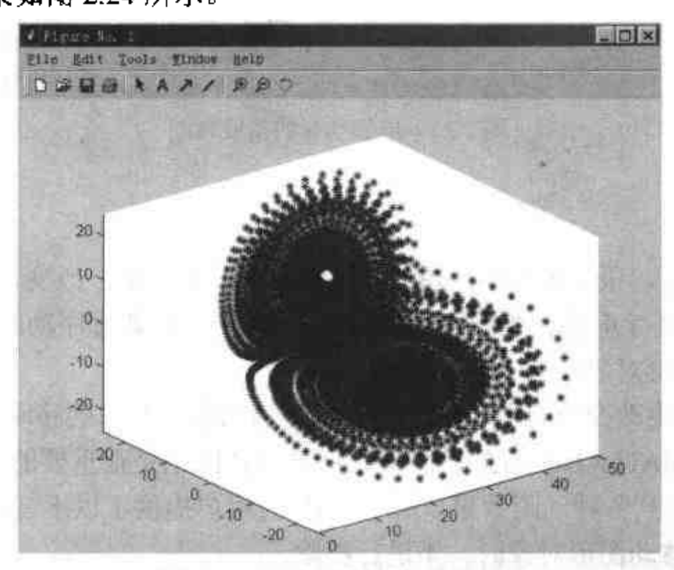


图 2.24 不进行擦除的程序动画

在上面的程序中由于我们把擦除模式设置为'none'，因而我们在图形窗口中看到由于点的移动所留下的轨迹。

如果将擦除模式设置为'background'，那么我们看到的只是一个点在坐标系中的运动所形成的动画，且动画播放过程中的一个画面如图 2.25 所示。

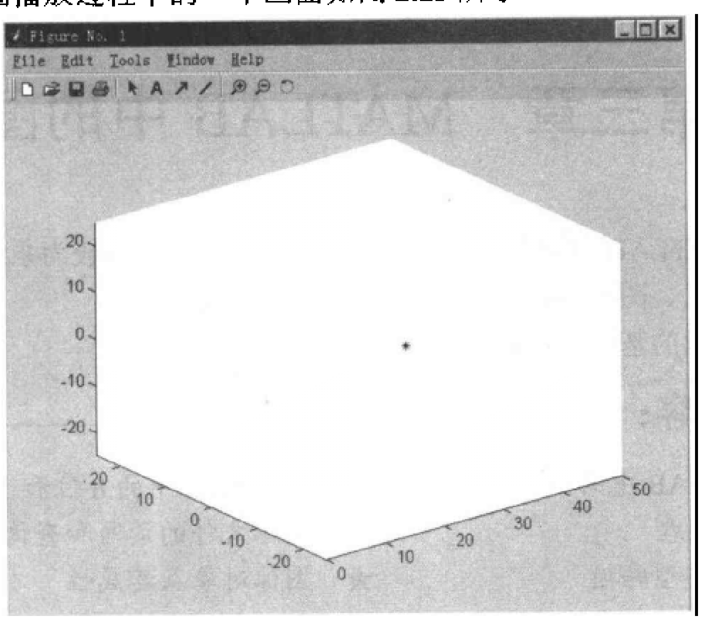


图 2.25 进行擦除的程序动画

第三章 MATLAB 中的图像

本章将针对 MATLAB 中有关图像的一些最基本的知识进行较为系统的介绍，这些知识包括图像的文件格式、图像的类型以及图像的存储类型等等。

本章为后续各章的基础，建议读者仔细阅读。

本章主要内容：

★ MATLAB 中的图像文件格式
★ 图像类型
★ 图像类型转换
★ MATLAB 中的 8 位和 16 位图像
★ 图像文件的读写和查询
★ 图像对象及其属性

3.1 MATLAB 中的图像文件格式

MATLAB 支持以下几种图像文件格式。

(1) PCX(Windows Paintbrush)格式：可处理 1、4、8、16、24 位等图像数据。文件内容包括文件头(128 字节)、图像数据和扩展颜色映射表数据。

(2) BMP(Windows Bitmap)格式：有 1、4、8、24 位非压缩图像，8 位 RLE(Run.length Encoded)图像。文件内容包括文件头(一个 BITMAP FILEHEADER 数据结构)、位图信息数据块(位图信息头 BITMAP INFOHEADER 和一个颜色表)和图像数据。

(3) HDF(Hierarchial Data Format)格式：有 8 位、24 位光栅图像数据集。

(4) JPEG(Joint Photographic Experts Group)格式：是一种称为联合图像专家组的图像压缩格式。

(5) TIFF(Tagged Image File Format)格式：处理 1、4、8、24 位非压缩图像，1、4、8、24 位 packbit 压缩图像，1 位 CCITT 压缩图像等。文件内容包括文件头、参数指针表与参数域、参数数据表和图像数据四部分。

(6) XWD(X Windows Dump)格式：1、8 位 Zpixmaps，XYBitmaps，1 位 XYPixmaps。

(7) PNG(Portable Network Graphics)格式。

3.2 图 像 类 型

在 MATLAB 中，一幅图像可能包含一个数据矩阵，也可能包含一个颜色映射表矩阵。

除了基本的图像类型外，MATLAB 还支持由多帧图像组成的图像序列。下面具体介绍 MATLAB 如何描述这几种图像。

3.2.1 索引图像

索引图像包括一个数据矩阵 X，一个颜色映像矩阵 Map。其中 Map 是一个包含三列、若干行的数据阵列，其每一个元素的值均为[0，1]之间的双精度浮点型数据。Map 矩阵的每一行分别表示红色、绿色和蓝色的颜色值。在 MATLAB 中，索引图像是从像素值到颜色映射表值的“直接映射”。像素颜色由数据矩阵 X 作为索引指向矩阵 Map 进行索引。例如，值 1 指向矩阵 Map 中的第一行，2 指向第二行，依此类推。

颜色映射表通常和索引图像存在一起。当用户在调用函数 imread 时，MATLAB 自动将颜色映射表与图像同时加载。在 MATLAB 中可以选择所需要的颜色映射表，而不必局限于使用默认的颜色映射表。我们可以使用属性 CDataMapping 来选取其它的颜色映射表，包括用户自定义的颜色映射表。

图 3.1 显示了索引图像的结构。该图像中的像素用整数类型表示，指向颜色映射表中的特定的颜色值。

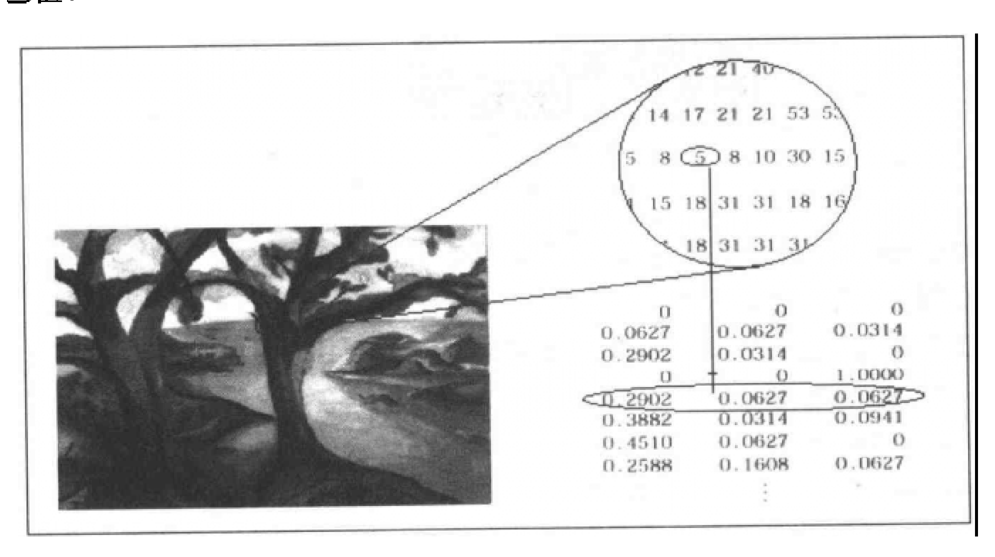


图 3.1 索引图像的结构

需要说明的是，图像矩阵与颜色映射表之间的关系取决于图像数据矩阵的类型。如果图像数据矩阵是双精度的数据类型，则值 1 指向矩阵 Map 中的第一行，值 2 指向第二行，依此类推；如果图像矩阵是 8 位无符号的整数类型或 16 位无符号的整数类型，则由于存在一个偏移量，因而使值 0 指向矩阵 Map 中的第一行，值 1 指向第二行，依此类推。在图 3.1 中，图像矩阵用的是双精度型，无偏移量，所以值 5 指向颜色映射表中的第 5 行。

例 3.2.1 显示一幅索引图像。

```
[X,map]=imread('canoe.tif');
image(X);
colormap(map);
```

3.2.2 灰度图像

在 MATLAB 中，一幅灰度图像是一个数据矩阵 I，I 中的数据均代表了在一定范围内的颜色灰度值。MATLAB 把灰度图像存储为一个数据矩阵，该数据矩阵中的元素分别代表了图像中的像素。矩阵中的元素可以是双精度的浮点类型、8 位或 16 位无符号的整数类型。大多数情况下，灰度图像很少和颜色映射表一起保存。但是在显示灰度图像时，MATLAB 仍然在后台使用系统预定义的默认灰度颜色映射表。

图 3.2 显示了灰度图像的结构。

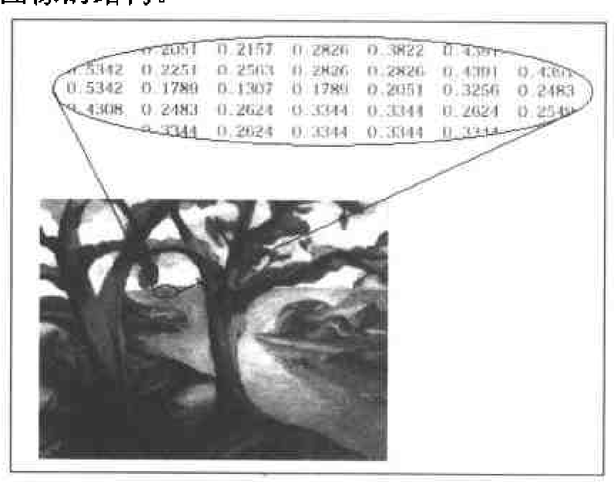


图 3.2　灰度图像的结构

在 MATLAB 中，可以调用图像色彩缩放函数 imagesc 来预先对图像进行色彩处理，再转换成灰度图以调整灰度的深浅。

例 3.2.2　用 imagesc 函数及 colormap 函数显示一幅灰度图像。

```
I=imread('moon.tif ');
imagesc(I,[0   256]);
colormap(gray);                 %原图像经灰度缩放后显示的灰度图像
```

imagesc 函数中的第二个参数确定色彩(灰度)范围。色彩范围中的第一个值(通常是 0)对应于颜色映射表中的第一个颜色值，灰度范围中的第二个值(通常是 256)对应于颜色映射表中的最后一个颜色值。颜色映射表中其余中间的颜色值将按照线性关系对应到这个色彩范围上。

3.2.3 RGB 图像

RGB 图像，即真彩图像，在 MATLAB 中存储为 n×m×3 的数据矩阵。数组中的元素定义了图像中每一个像素的红、绿、蓝颜色值。需要指出的是，RGB 图像不使用 Windows 颜色映射表。像素的颜色由保存在像素位置上的红、绿、蓝的灰度值的组合来确定。图形文件格式把 RGB 图像存储为 24 位的图像，红、绿、蓝分别占 8 位，这样可以有 1000 多万种颜色(即 2^{24}=16 777 216 种颜色)。

MATLAB 的 RGB 数组可以是双精度型的浮点类型、8 位或 16 位无符号的整数类型。

在 RGB 的双精度型数组中，每一种颜色是用在 0 和 1 之间的数值表示。例如，颜色值为(0，0，0)时的像素，显示的是黑色；颜色值为(1，1，1)时的像素，显示的是白色。每一像素的三个颜色值保存在数组的第三维中。例如，像素(10，5)的红、绿、蓝颜色值分别保存在元素 RGB(10,5,1)、RGB(10,5,2)、RGB(10,5,3)中。

例 3.2.3 调用函数 image 来显示 RGB 图像。

```
RGB=imread('flowers.tif ');
image(RGB);
```

打开的真彩图像如图 3.3 所示。

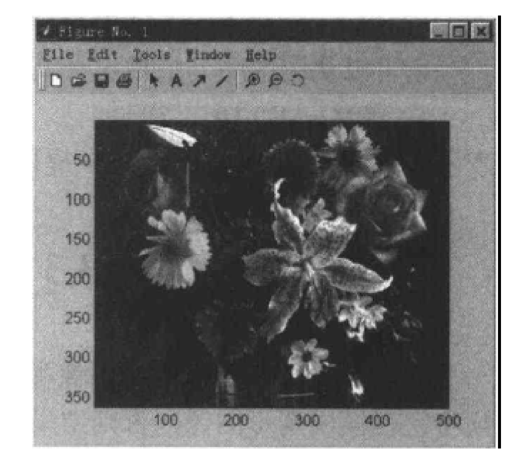


图 3.3 真彩图像

在上面的 RGB 图像中，要确定像素(12，9)的颜色，可以在命令行中键入

```
RGB(12,9,:)
```

回车，得到

```
ans(:,:,1) = 59
ans(:,:,2) = 55
ans(:,:,3) = 91
```

即像素(12，9)的 RGB 颜色为：59(红色)、55(绿色)、91(蓝色)。

如果 MATLAB 在一台不支持真彩显示的计算机上运行，则 MATLAB 用颜色近似和抖动来显示相近的真彩颜色。

3.2.4 二值图像

与灰度图像相同，二值图像只需要一个数据矩阵，每个像素只取两个灰度值。二值图像可以采用 uint8 或 double 类型存储，工具箱中以二值图像作为返回结果的函数都使用 uint8 类型。

图 3.4 是一幅二值图像的结构。

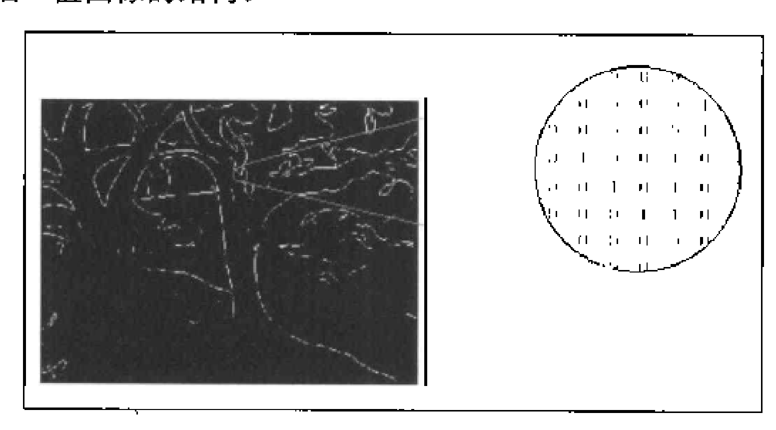


图 3.4 二值图像的结构

3.2.5 图像序列

MATLAB 的图像处理工具箱还支持将多帧图像连接成图像序列。图像序列是一个四维的数组，图像帧的序号在图像的长、宽、颜色深度之后构成第四维。比如一个包含了 5 幅

400×300 真彩图像的序列，其大小为 400×300×3×5。

要将分散的图像合并成图像序列，可以使用 MATLAB 的 cat 函数，前提是各图像的尺寸必须相同，如果是索引图像，颜色映射表也必须是一样的。

比如，要将 A1、A2、A3、A4、A5 五幅图像合并成一个图像序列 A，MATLAB 命令为

```
A=cat(4,A1,A2,A3,A4,A5)
```

也可以从图像序列中抽出一帧，比如代码：

```
FRM3=MULTI(:,:,:,3)
```

表示将序列 MULTI 中的第三帧提取出来赋给矩阵 FRM3。

3.3 图像类型转换

许多图像处理工作都对图像类型有特定的要求。比如要对一幅索引图像滤波，首先必须将它转换成真彩图像，否则结果是毫无意义的。

在 MATLAB 中，各种图像类型之间的转换关系如图 3.5 所示。

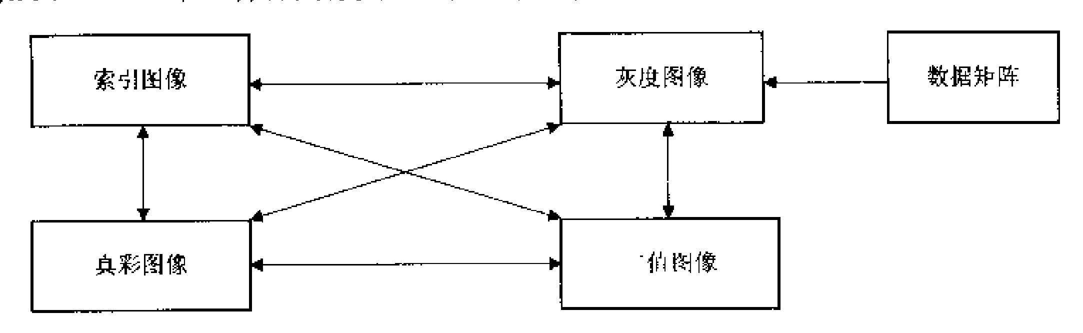


图 3.5 图像类型间的转换

MATLAB 的图像处理工具箱提供了许多图像类型转换函数，以实现上述的各种转换。下面具体介绍这些函数的用法和功能。

3.3.1 dither 函数

该函数通过颜色抖动(颜色抖动即改变边沿像素的颜色，使像素周围的颜色近似于原始图像的颜色，从而以空间分辨率来换取颜色分辨率)来增加输出图像的颜色分辨率，从而实现转换图像。

该函数的调用格式如下：

```
X = dither(RGB,map)
BW = dither(I)
```

其中：

X = dither(RGB,map)表示将真彩图像 RGB 按指定的颜色映射表 map 抖动成索引图像 X。

BW = dither(I)表示将灰度图像 I 抖动成二值图像 BW。

另外，输入图像可以是 double 或 uint8 类型，输出图像如果是二值图像或颜色种类不超

过 256 的索引图像，则是 uint8 类型，否则为 double 型。

例 3.3.1 将灰度图像 rice.tif 转换为二值图像。

```
I=imread('rice.tif');
BW=dither(I);
imshow(BW);
```

原图及转换后的结果如图 3.6 所示。

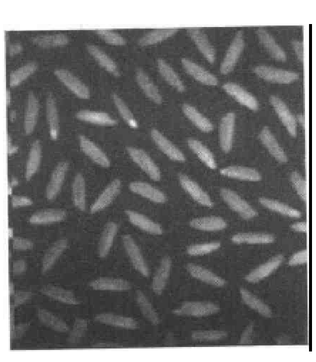
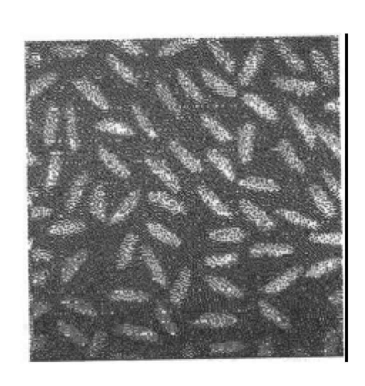



图 3.6 灰度图像转换成二值图像

3.3.2 gray2ind 函数

该函数的功能是将灰度图像转换成索引图像。其调用格式如下：

[X, map] = gray2ind(I,n)

其中：

[X, map] = gray2ind(I,n)表示按指定的灰度级数 n 将灰度图像 I 转换成索引图像 X，n 的缺省值为 64。

另外，输入图像 I 可以是 double 类型、uint8 类型或 uint16 类型。如果颜色映射表的长度小于等于 256，则输出图像是 uint8 类型，否则为 double 类型。

例 3.3.2 把一幅灰度图像转换为一幅索引图像。

```
I=imread('rice.tif');
[X,map]=gray2ind(I,6);
imshow(X,map);
```

原图与转换后的图像的比较如图 3.7 所示。

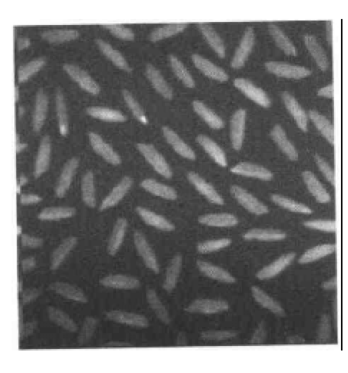
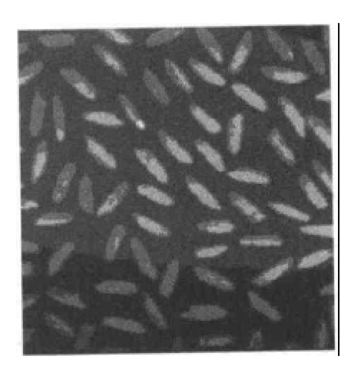



图 3.7 灰度图像转换成索引图像

3.3.3 grayslice 函数

该函数的功能是通过设定阈值将灰度图像转换成索引图像。其调用格式如下：

X = grayslice(I,n)

X = grayslice(I,v)

其中：

X = grayslice(I,n) 将灰度图像 I 均匀量化为 n 个等级，然后转换为伪彩色图像 X。

X = grayslice(I,v)按指定的阈值向量 v(其每一个元素都在 0 和 1 之间)对图像 I 的值域进行划分，而后转换成索引图像 X。

另外，输入图像 I 可以是 double 或 uint8 类型。如果阈值数量小于 256，则返回图像 X 的数据类型是 uint8，X 的值域为[0, n]或[0, length(v)]；否则，返回图像 X 为 double 类型，值域为[1, n+1]或[1, length(v)+1]。

例3.3.3 将一幅灰度图像转换成索引图像。

```
I = imread('ngc4024m.tif');
X = grayslice(I,16);
imshow(I)
figure, imshow(X,hot(16))
```

原图与转换后的图像的比较结果见图 3.8。

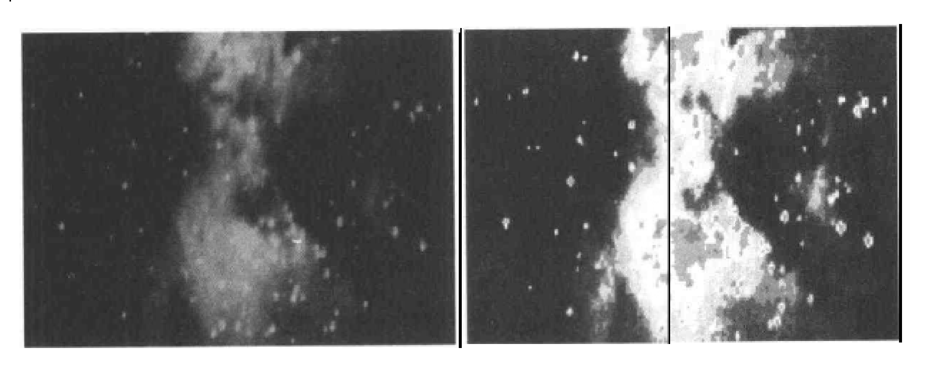


图 3.8 灰度图像转换成索引图像

3.3.4 im2bw 函数

该函数的功能是通过设置亮度阈值将真彩、索引、灰度图像转换成二值图像。该函数的调用格式如下：

BW = im2bw(I,level)

BW = im2bw(X map,level)

BW = im2bw(RGB,level)

其中，BW = im2bw(I,level) 、BW = im2bw(X,map,level)和 BW = im2bw(RGB,level)分别表示将灰度图像、索引图像和真彩图像二值化为图像 BW。level 是归一化的阈值，取值在[0, 1]之间。

另外，输入图像可以是 double 或 uint8 类型，输出图像为 uint8 类型。

例 3.3.4 将一幅索引图像转换成二值图像。

```
load trees
BW = im2bw(X,map,0.4);
imshow(X,map)
figure, imshow(BW)
```

原图与转换后的图像的比较结果见图 3.9。




图 3.9 索引图像转换成二值图像

3.3.5 ind2gray 函数

该函数的功能是将索引图像转换成灰度图像。其调用格式如下：

I = ind2gray(X,map)

需要说明的是，输入图像可以是 double 或 uint8 类型，输出图像为 double 类型。

例 3.3.5 将一幅索引图像转换成一幅灰度图像。

```
load trees
I = ind2gray(X,map);
imshow(X,map)
figure,imshow(I)
```

原图与转换后的图像的比较结果如图 3.10 所示。




图 3.10 索引图像转换成灰度图像

3.3.6 ind2rgb 函数

该函数的功能是将索引图像转换成真彩图像。其调用格式如下：

RGB = ind2rgb(X,map)

需要说明的是，输入图像 X 可以是 double 或 uint8 类型，输出图像 RGB 为 double 类型。

3.3.7 mat2gray 函数

该函数的功能是将一个数据矩阵转换成一幅灰度图像。其调用格式如下：

I = mat2gray(A,[amin amax])

I = mat2gray(A)

其中，I = mat2gray(A,[amin amax])表示按指定的取值区间[amin amax]将数据矩阵 A 转化为灰度图像 I，amin 对应灰度 0(最暗)，amax 对应 1(最亮)。如果不指定区间[amin amax]，则 Matlab 自动将 A 阵中的最小元设为 amin，最大元设为 amax。

另外，输入矩阵 A 与输出图像 I 都为 double 类型。

例 3.3.6 Sobel 算子对图像滤波，将滤波得到的数据矩阵转换为灰度图像。

```
I = imread('rice.tif');
J = filter2(fspecial('sobel'),I);
K = mat2gray(J);
imshow(I)
figure, imshow(K)
```

原图与转换后的图像的比较结果如图 3.11 所示。

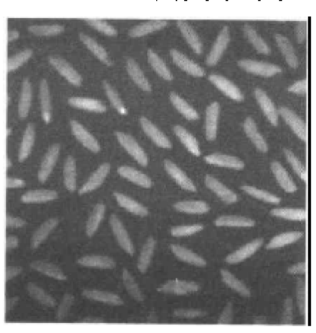
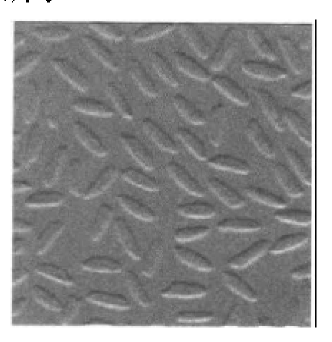



图 3.11 数据矩阵转换成灰度图像

3.3.8 rgb2gray 函数

该函数的功能有两个：第一个功能是将一幅真彩图像转换成一幅灰度图像；第二个功能是把一幅索引图像的彩色颜色映射表(即颜色映射表)转换为灰度颜色映射表。其调用格式如下：

I = rgb2gray(RGB)

newmap = rgb2gray(map)

其中：

I = rgb2gray(RGB)表示将真彩图像 RGB 转换成灰度图像 I；

newmap = rgb2gray(map)表示将彩色颜色映射表 map 转化成灰度颜色映射表。

需要注意的是，如果输入的是真彩图像，则可以是 uint8 或 double 类型，输出图像 I 与输入图像类型相同。如果输入的是颜色映射表，则输入、输出都是 double 类型。

例 3.3.7 将一幅彩色索引图像转换为一幅颜色映射表为灰度颜色映射表的索引图像(为非彩色图像)。

```
[X,map]=imread('canoe.tif');
newmap=rgb2gray(map);
imshow(X,newmap);
```

3.3.9 rgb2ind 函数

该函数的功能是将索引图像转换成真彩图像。其调用格式如下：

RGB = rgb2ind (X,map)

其中，X 和 map 分别代表原输入索引图像的数据矩阵和颜色映射表。

另外，输入图像 X 可以是 double 或 uint8 类型，输出图像 RGB 为 double 类型。

3.4 MATLAB 中的 8 位和 16 位图像

MATLAB 中最基本的数据结构是数组。在 MATLAB 中，大多数图像用二维数组(矩阵)存储，矩阵中的一个元素对应于所要显示图像的一个像素。例如，一个由 100 行和 200 列的不同颜色的点组成的图像可以用一个 100×200 的矩阵来存储。也有一些图像，如 RGB 图像，需要用三维数组来存储，第一维表示红色像素的深度值，第二维表示绿色像素的深度值，第三维表示蓝色像素的深度值。

这意味着 MATLAB 强大的矩阵运算功能完全可以应用于图像，那些适用于矩阵运算的语法对 MATLAB 中的数字图像同样适用。

在缺省的情况下，MATLAB 将图像中的数据存储为双精度类型(double)，即 64 位的浮点数。这种存储方法的优点在于使用中不需要数据类型转换，因为几乎所有的 MATLAB 及其工具箱函数都可使用 double 作为参数类型。然而，对于图像存储来说，用 64 bit 来表示图像数据会导致巨大的存储量，所以 MATLAB 还支持图像数据的另一种类型——无符号整型(uint8),即图像矩阵中的每个数据占用一个字节。MATLAB 及工具箱中的大多数操作及函数(比如最基本的矩阵相加)都不支持 uint8 类型。uint8 的优势仅仅在于节省存储空间，在涉及运算时要将图像数据转换成 double 型。

在处理图像时，MATLAB 通常使用 64 位的双精度浮点类型。但是，为了减少内存需求，提高系统运行效率，MATLAB 还提供了两个重要的数字类型 uint8 和 uint16，用以支持 8 位和 16 位的无符号整数类型。在 MATLAB 中，数据矩阵中包含 uint8 数字类型的图像称之为 8 位图；同理，数据矩阵中包含 uint16 数字类型的图像称为 16 位图。

利用 MATLAB 提供的 image 函数，可以直接显示 8 位图像或 16 位图像，而不必将其转换为双精度浮点类型。然而，当图像是 uint8 或 uint16 类型时，MATLAB 对矩阵值的解释有所不同，这主要依赖于图像的具体类型

3.4.1 8 位和 16 位索引图像

如果图像数据矩阵的类型是 uint8 或 uint16，则其值在作用于颜色映射表的索引之前，必须进行值为 1 的偏移，即值 0 指向矩阵 map 的第一行，值 1 指向第二行，依此类推。并且因为 image 函数自动提供了这种偏移，所以不管图像数据矩阵是双精度浮点类型，还是

uint8 或 uint16，显示的方法均相同。

但是，当用户在进行类型转换时，由于偏移量的存在，必须将 uint8 或 uint16 的数据加 1，然后才能将其转换为双精度浮点类型。例如：

```
X64=double(X8)+1;
```

或

```
X64=double(X16)+1;
```

相反，若将双精度浮点类型转换为 uint8 或 uint16 类型的数据时，在转换之前，必须将其减去 1。例如：

```
X8=uint8(X64-1);
```

或

```
X16=uint16(X64-1);
```

3.4.2 8 位和 16 位灰度图像

在 MATLAB 中，双精度浮点类型的图像数组中的数值的取值范围通常为[0，1]，而 8 位和 16 位无符号整数类型的图像的取值范围分别为[0，255]和[0，65535]。

例 3.4.1 显示一个 8 位的灰度图。

```
I=imread('rice.tif ');
Imagesc(I,[0 255]);
colormap(gray);
```

在 MATLAB 中，在将一个灰度图从双精度的浮点类型转换为 uint16 的 16 位无符号整数类型时，必须首先将其乘以 65 535。例如：

```
I16=uint16(round(I64*65535));
```

与此相反，在将一个灰度图从 uint16 的 16 位无符号整数类型转换为双精度的浮点类型时，必须首先除以 65 535。例如：

```
I64=double(I16)/65535;
```

3.4.3 8 位和 16 位 RGB 图像

8 位 RGB 图像的颜色数据是[0，255]之间的整数，而不是[0，1]之间的浮点值。所以，在 8 位 RGB 图像中，颜色值为(255，255，255)的像素显示为白色。不管 RGB 图像是何种类型，MATLAB 都通过以下的代码来显示，即

```
image(RGB);
```

将 RGB 图像从双精度的浮点类型转换为 uint8 无符号整数类型时，首先要乘以 255，即：

```
RGB8=uint8(round(RGB64*255));
```

相反，如果将 uint8 无符号整数类型的 RGB 图像转换为双精度的浮点类型时，首先要除以 255，即：

```
RGB64=double(RGB8)/255;
```

另外，如果将 RGB 图像从双精度的浮点类型转换为 uint16 无符号整数类型时，必须乘以 65 535，即：

```
RGB16=uint16(round(RGB64*65535));
```

同样，如果将 uint16 无符号整数类型的 RGB 图像转换为双精度浮点类型时，首先要除以 65 535，即：

```
RGB64=double(RGB16)/65535;
```

3.5 图像文件的读写和查询

在对一幅图像文件进行处理时，先抛开具体的处理过程不说，首先要熟悉以下的几个操作：

(1) 查询该图像的相关信息；

(2) 读取该图像的相关数据；

(3) 将处理过的图像数据保存到另一个图像文件中。

针对上述的几个操作，MATLAB 都给用户提供了相关的函数。下面详细介绍这几个相关的函数。

3.5.1 图像文件信息的查询

MATLAB 中，用于图像文件信息查询的是 imfinfo 函数。其格式如下：

```
info = imfinfo('文件名',文件格式)
info = imfinfo('文件名')
```

由该函数获取的信息随文件类型的不同而不同，但至少包含以下内容：

(1) '文件名'(文件名)；

(2) FileModDate(文件最后一次的修改时间)；

(3) FileSize(文件的大小，单位为字节)；

(4) Format(文件格式)；

(5) FormatVersion(文件格式的版本号)；

(6) Width(图像的宽度，单位为像素)；

(7) Height(图像的高度，单位为像素)；

(8) BitDepth(每个像素的位数)；

(9) ColorType(图像类型，即是 RGB 图像、灰度图像还是索引图像)。

例如，在 MATLAB 的命令行中输入下面的代码，以查询文件 rice.tif 的信息：

```
info=imfinfo('rice.tif ')
```

执行后，得到的结果如下：

```
info =
        '文件名': 'rice.tif '
        FileModDate: '25.Oct.1996 22:11:58'
        FileSize: 65966
        Format: 'tif'
        FormatVersion: [ ]
```

```
Width: 256
Height: 256
BitDepth: 8
ColorType: 'grayscale'
FormatSignature: [73 73 42 0]
ByteOrder: 'little.endian'
NewSubfileType: 0
BitsPerSample: 8
Compression: 'Uncompressed'
PhotometricInterpretation: 'BlackIsZero'
StripOffsets: [8x1 double]
SamplesPerPixel: 1
RowsPerStrip: 32
StripByteCounts: [8x1 double]
XResolution: 72
YResolution: 72
ResolutionUint: 'Inch'
Colormap: [ ]
PlanarConfiguration: 'Chunky'
TileWidth: [ ]
TileLength: [ ]
TileOffsets: [ ]
TileByteCounts: [ ]
Orientation: 1
FillOrder: 1
GrayResponseUint: 0.0100
MaxSampleValue: 255
MinSampleValue: 0
Thresholding: 1
ImageDescription: [1x166 char]
```

3.5.2 图像文件的读取

MATLAB 中利用函数 imread 来实现图像文件的读取操作。其格式如下：

A = imread('文件名',文件格式)等价于A = imread('文件名.文件格式')

[X, map] = imread('文件名',文件格式)

[...] = imread('文件名')

[...] = imread(...,idx) (只适用于*.tif格式)

[...] = imread(...,ref) (只适用于*.HDF 格式)

其中：

[...] = imread(...,idx) 用于读取多帧 TIFF 文件中的一帧；

[...] = imread(...,ref) 用于读取多帧 HDF 文件中的一帧。

通常情况下，我们通过 imread 函数读取的大多数图像都是 8 位的。当把这些图像加载到内存中时，MATLAB 就将其存储在类 uint8 中。

此外，MATLAB 还支持 16 位的 PNG 和 TIFF 图像，当用户读取这类图像时，MATLAB 就将其存储在类 uint16 中。而对于索引图像来说，即使图像矩阵的本身为 uint8 或 uint16，imread 函数仍将颜色映射表读取并存储在一个双精度浮点类型的矩阵中。

3.5.3 图像文件的保存

在 MATLAB 中利用函数 imwrite 来实现图像文件的保存操作。其格式如下：

```
imwrite(A, '文件名',文件格式)
imwrite(X,map, '文件名',文件格式)
imwrite(...,'文件名')
imwrite(...,Parameter,Value,...)
```

其中：

文件格式可以取以下的值："bmp"、"hdf"、"jpg"、"pcx"、"png"、"tif" 和 "xwd"；

imwrite(...,'文件名')中的文件名必须带合法的扩展名；

imwrite(...,Parameter,Value,...)可以让用户控制 HDF、JPEG、TIFF 三种图像文件的输出特性。

对于 HDF 文件，参数说明见表 3.1。

表 3.1 对于 HDF 文件有效的参数的说明

Parameter	可 选 值	缺 省 值
'Compression'	'none'、'rle'、'jpeg'。其中，'rle'只对灰度图像和索引图像有效；'jpeg'只对灰度图像和 RGB 图像有效	'rle'
'Quality'	取值为 0 到 100。只有 Compression 为'jpeg'时才有效，且其值越大，压缩质量越高，但文件也越大	75
'WriteMode'	'overwrite'或'append'	'overwrite'

对于 JPEG 文件，参数说明见表 3.2。

表 3.2 对于 jpeg 文件有效的参数的说明

Parameter	可 选 值	缺 省 值
'Quality'	取值为 0 到 100，值越大，压缩质量越高，但文件也越大	75

对于 TIFF 文件，参数说明见表 3.3。

表 3.3 对于 TIFF 文件有效的参数的说明

Parameter	可 选 值	缺 省 值
'Compression'	'none'、'packbits'、'ccitt'。其中'ccitt'只对二值图像有效	对于二值图像是'none'；对于其它图像是'packbits'
'Description'	任意字符串，填充由函数 imfinfo 返回的 ImageDescription 域	空
'Resolution'	一个标量值，用来定义输出文件的 x 和 y 方向的分辨率	72

例 3.5.1 将一幅 tif 格式的图像保存为一幅 hdf 格式的图像。

```
[X,map]=imread('canoe.tif');
imwrite(X,map,'canoe.hdf','Compression','none',...
    'WriteMode','append')
```

另外，需要注意的是，当利用 imwrite 函数保存图像时，MATLAB 默认的保存方式是将其简化为 uint8 的数据类型。与读取图像文件类型类似，MATLAB 在文件保存时还支持 16 位的 PNG 和 TIFF 图像。所以，当用户保存这类文件时，MATLAB 就将其存储在 uint16 中。

3.6 图像对象及其属性

在 MATLAB 中，可以调用 image 函数和 imagesc 函数来创建图像对象。与线条对象、片块对象、表面对象以及文本对象类似，图像对象也是坐标轴对象的子对象。另外，与所有的句柄图形对象一样，图像对象也包含许多用户可以设置的以调整其屏幕显示外观的属性。其中最重要的属性包括 CData 属性、CDataMapping 属性、XData 属性和 YData 属性等。

3.6.1 图像对象的 CData 属性

图像对象的属性 CData 包含一个数据阵列。下面的代码中，h 表示由 image 函数创建的图像对象的句柄；X 和 Y 代表同一个矩阵。

```
H=image(X);
colormap(map);
Y=get(h, 'CData');
```

其中，属性 CData 数据阵列的维数决定了 MATLAB 是使用颜色映射表还是使用 RGB 真彩的方式显示该图像。如果 CData 属性的数据阵列是二维的，则图像显示为索引图像或灰度图像。但不管是何种方式，MATLAB 均使用颜色映射表。如果 CData 属性的数据阵列是三维的，形如 m×n×3，那么 MATLAB 就将其显示为真彩图像。

3.6.2 图像对象的 CDataMapping 属性

图像对象的 CDataMapping 属性决定了图像是索引图像还是灰度图像。如果将该属性设置为“direct”，则该图像显示为索引图像，在这种情况下，CData 属性中的数据阵列直接作为当前图形窗口的颜色映射表中的颜色索引。在 MATLAB 中，只要我们在调用 image 函数时只使用了一个参数，则系统自动将 CDataMapping 属性的值设置为“direct”。

例如，我们在命令行中输入下面的代码：

```
h=image(X);
colormap(map);
get(h, 'CDataMapping');
```

回车执行后，得到下面的结果：

```
ans=
    direct
```

另外，如果将 CDataMapping 属性的值设置为“scaled”，则 MATLAB 将该图像对象显示为灰度图像。在这种情况下，MATLAB 将 CData 属性的数据线性缩放，生成颜色映射表的索引，其中缩放因子由坐标轴对象的 CLim 属性确定。

在 MATLAB 中，只要利用 imagesc 函数创建图像对象，系统就会自动将 CDataMapping 属性的值设置为“scaled”，并且同时调整其父坐标轴对象的 CLim 属性。

例如，在命令行中输入下面的代码：

```
h=imagesc(I,[0 1]);
colormap(map);
get(h, 'CDataMapping');
```

回车执行后，得到下面的结果：

```
ans=
    scaled
```

输入下面的代码：

```
get(gca, 'CLim');
```

回车执行后，得到下面的结果：

```
ans=
    [0 1]
```

3.6.3　图像对象的 XData 和 YData 属性

图像对象的 XData 和 YData 属性决定了图像的坐标系统。对于一个数据矩阵为 m×n 的图像对象来说，其 XData 属性的默认值是[1 n]，YData 属性的默认值是[1 m]。这样的设置意义如下：

(1) 图像数据矩阵的左边的列具有一个值为 1 的 X 坐标。

(2) 图像数据矩阵的右边的列具有一个值为 n 的 X 坐标。

(3) 图像数据矩阵的上边的行具有一个值为 1 的 Y 坐标。

(4) 图像数据矩阵的下边的行具有一个值为 m 的 Y 坐标。

例 3.6.1　在默认的坐标系中创建一个矩阵图像对象。

```
X=[1 2 3 4;5 6 7 8;1 2 3 4];
h=image(X);
colormap(map)
xlabel('x轴');
ylabel('y轴');
```

生成如图 3.12 所示的矩阵图像对象。

这样生成的图像对象的 XData 和 YData 属性具有默认的值。在命令行中输入下面的代码：

```
get(h, 'XData');
```

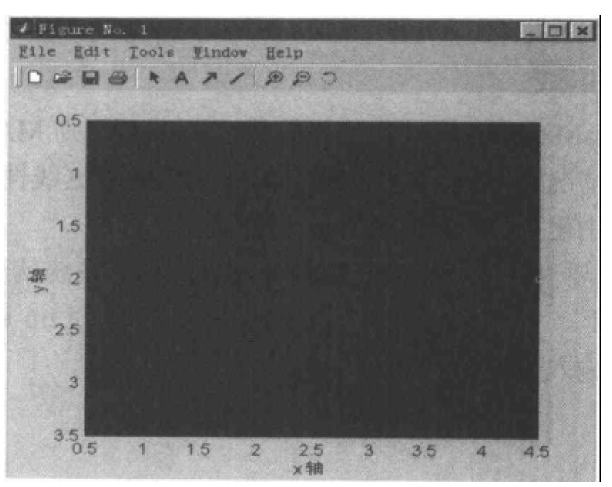


图 3.12 默认坐标系下的图像显示

回车执行后，得到如下结果：

```
ans=
    1    4
```

同样，代码：

```
get(h, 'YData');
```

执行的结果如下：

```
ans=
    1    3
```

另外，用户还可以定义自己的坐标系来覆盖系统的默认的设置。

例 3.6.2 在自定义的坐标系中创建一个矩阵图像对象。

```
X=[1 2 3 4; 5 6 7 8;1 2 3 4];
image(X,'XData',[1 2],'YData',[2 4]);
colormap(map);
xlabel('x轴');
ylabel('y轴');
```

定义一个新的坐标系，其结果如图 3.13 所示，观察两图的坐标并比较。

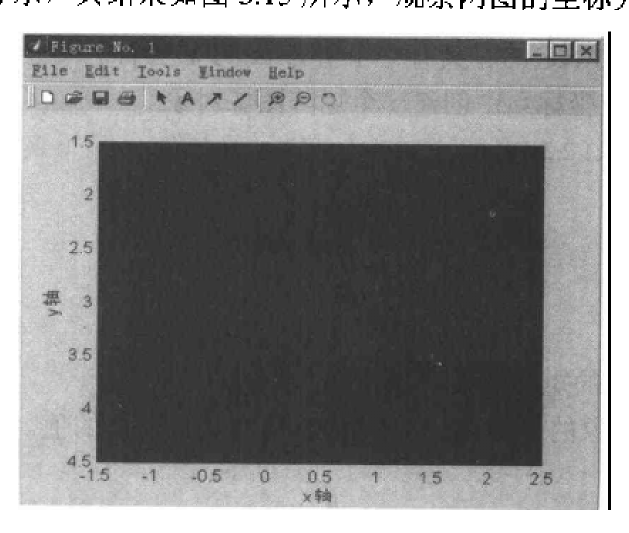


图 3.13 自定义坐标系下的图像显示

第四章 MATLAB 中的图像显示技术

MATLAB 的图像处理工具箱提供了多种图像显示和颜色处理技术。本章将具体介绍 MATLAB 中用于图像显示以及用于颜色模型转换的各个函数的具体用法。

本章主要内容：

★ 标准的图像显示技术
★ 特殊图像显示技术
★ MATLAB 中的颜色模型

4.1 标准的图像显示技术

由第三章已经知道，可以调用 image 函数和 imagesc 函数来显示一幅图像。在 MATLAB 的图像处理工具箱中，还提供了一个应用很广泛的图像显示函数，即 imshow 函数。与 image 函数和 imagesc 函数类似，imshow 函数也创建句柄图形图像对象。此外，imshow 函数也可以自动设置各种句柄图形属性和图像特征，以优化显示效果。

4.1.1 imshow 函数

当用户调用 imshow 函数显示一幅图像时，该函数将自动设置图像窗口、坐标轴和图像属性。这些自动设置的属性具体包括图像对象的 CData 属性和 CDataMapping 属性、坐标轴对象的 CLim 属性、图像窗口对象的 Colormap 属性。

另外，根据用户使用参数的不同，imshow 函数在调用时除了完成前面提到的属性设置外，还可以完成以下的操作：

(1) 设置其它的图形窗口对象的属性和坐标轴对象的属性以优化显示效果，如可以设置隐藏坐标轴及其标示等。

(2) 包含和隐藏图像边框。

(3) 调用 truesize 函数以显示没有彩色渐变效果的图像。

imshow 函数的调用格式如下：

```
imshow(I,n)
imshow(I,[low high])
```

```
imshow(BW)
imshow(X,map)
imshow(RGB)
imshow(...,display_option)
imshow(x,y,A,...)
imshow filename
```

每一种格式的具体用途将在下面详细介绍。

4.1.2 显示灰度图像

调用 imshow 函数显示灰度图像的格式如下：

```
imshow(I,n)
imshow(I,[low high])
```

其中：

I 代表所显示的灰度图像的数据矩阵；

n 为整数，代表所要显示图像的灰度等级数，缺省值为 256；

[low high]为图像数据的值域。需要注意的是，在很多情况下，经过处理的图像数据的值域都会发生变化。比如，对一幅 double 型的灰度图像滤波后，图像数据的值域已不在[0,1]中了，如果还用前面的显示方法，则得不到正确的结果。如果清楚地知道数据的值域[low high]，可以调用 imshow(I,[low high])；否则可以用空向量为参数，即 imshow(I,[ ])。

例 4.1.1 在一定灰度范围内显示灰度图像。

```
I=imread('rice.tif');
imshow(I,[100 200])
```

可以得到如图 4.1 所示的结果。

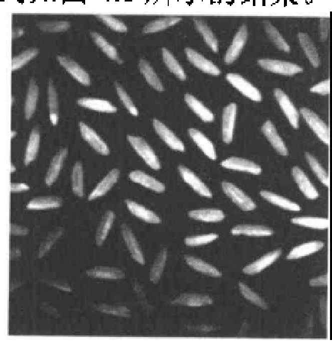


图 4.1 定义了灰度范围的灰度图像

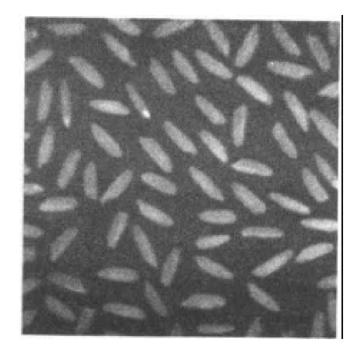


图 4.2 定义了灰度等级的灰度图像

例 4.1.2 按照指定的灰度等级显示灰度图像。

```
I=imread('rice.tif ');
imshow(I,20)
```

可以得到如图 4.2 所示的结果。

4.1.3 显示二值图像

调用 imshow 函数显示二值图像的格式如下：

imshow(BW)

需要说明的是，如果该二值图像矩阵属于 double 类，则 imshow 函数将其视为灰度图来对待，同时将 CDataMapping 的属性值设置为 scaled，CLim 的属性值设置为[0 1]；Colormap 颜色映射表的属性值设置为灰度颜色映射表。所以，在这种情况下，图像数据矩阵中值 0 所对应的像素显示为黑色，而值 1 所对应的像素显示为白色。

例 4.1.3　显示一幅二值图像。

```
BW=imread('circles.tif');
imshow(BW)
```

得到如图 4.3 所示的结果。

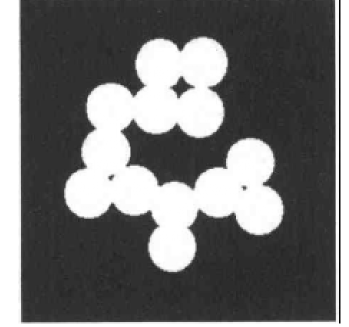


图 4.3　二值图像

4.1.4　显示索引图像

调用 imshow 函数显示索引图像的格式如下：

imshow(X,map)

其中：

X 代表索引图像的数据矩阵；

map 为颜色映射表。

另外，显示索引图像时，imshow 函数将同时设置下面的一些用来控制显示颜色的句柄图形的属性：

(1) 将图像的 CData 属性值设置为 X 矩阵中的数据；

(2) 将图像的 CDataMapping 属性值设置为 direct；

(3) 使坐标轴对象的属性失效，因为图像的 CDataMapping 属性值已经被设置为 direct；

(4) 将图形窗口对象的 Colormap 属性值设置为 map 矩阵中的数据。

4.1.5　显示真彩图像

调用 imshow 函数显示真彩图像的格式如下：

imshow(RGB)

其中 RGB 为代表该真彩图像的 $m\times n\times 3$ 的数据阵列。

4.1.6　显示图形文件中的图像

前面介绍的几种用 imshow 函数显示图像的方法中，均调用了所显示图像的对象数据。如果所显示图像被保存在可以通过 imread 函数读取的图像文件中，则 imshow 函数可以通过下面的格式直接将其显示出来：

imshow　文件名；

需要注意的是，该文件名必须带有合法的扩展名（即指明文件格式），且该图像文件必须保存在当前目录下，或在 MATLAB 默认的目录下。

例 4.1.4　显示一幅在当前目录下的 .jpg 格式的图像。

```
imshow   pic005.jpg;
```

显示的结果如图 4.4 所示。

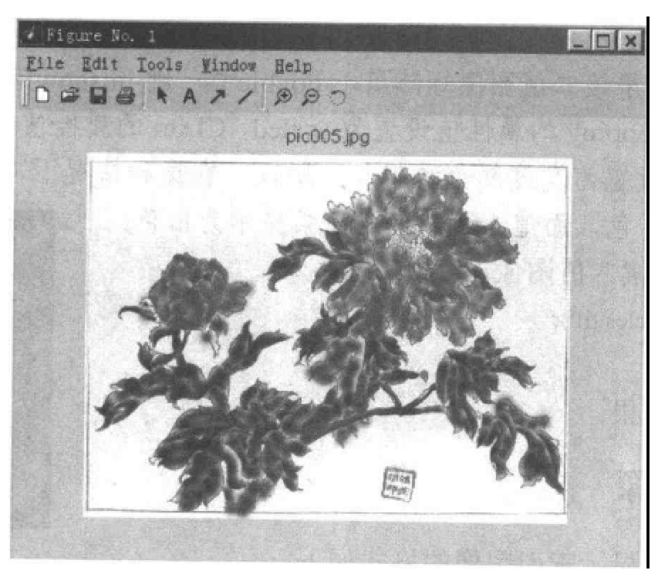


图 4.4 显示一幅图像文件中的图像

4.2 特殊图像显示技术

除了前面已经介绍的 imshow 函数以外，MATLAB 的图像处理工具箱还提供了一些可以实现特殊显示功能的函数。下面具体介绍这些函数的用法和功能。

4.2.1 添加颜色条

在 MATLAB 的图像显示中，可以利用 colorbar 函数将颜色条添加到坐标轴对象中。添加进来的颜色条用来指示图像中不同颜色所对应的数据值。

colorbar 函数的语法格式如下：

```
colorbar('vert')
colorbar('horiz')
colorbar(h)
colorbar
h = colorbar(...)
```

其中：

colorbar('vert')、colorbar('horiz')指定了颜色条的显示方式为垂直或水平，缺省值为垂直('vert')；

colorbar(h)表示将颜色条放在指定的坐标轴 h 上，h 为句柄；

h = colorbar(...)表示返回颜色条坐标轴的句柄。

例 4.2.1 添加了颜色条后的图像显示。

```
imshow pic005.jpg;
colorbar;
```

得到如图 4.5 所示的结果。

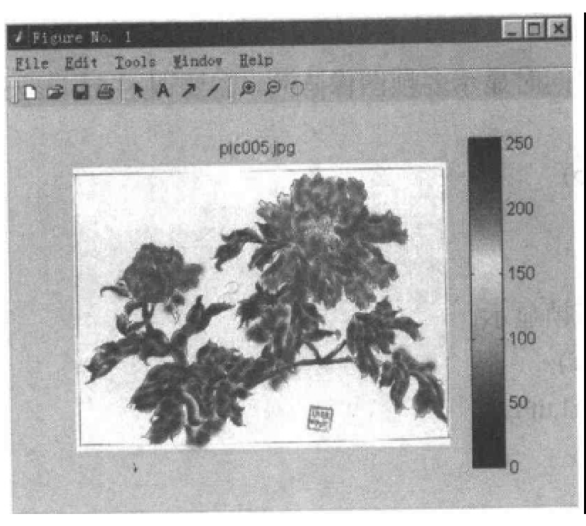


图 4.5 添加颜色条后的图像显示

4.2.2 显示多帧图像阵列

在 MATLAB 中，实现在多帧图像阵列中查看图像，有下面几种方式：

(1) 独立显示每一帧，调用imshow函数。

(2) 同时显示所有的帧，调用montage函数。

(3) 将多帧图像阵列转换为电影动画，调用immovie函数。

下面将具体介绍这几个函数的用法。

1. 单帧显示

下面通过一个例子来说明图像的单帧显示。

例 4.2.2 将两幅灰度图像合并成一个具有两帧的图像阵列，然后再调用 imshow 函数来显示第一帧图像：

```
A1=imread('rice.tif');
A2=imread('testpat1.tif');
A=cat(3,A1,A2);              %用cat函数实现矩阵的合并
imshow(A(:,:,1))             %1代表第一帧
```

得到如图 4.6 所示的结果。

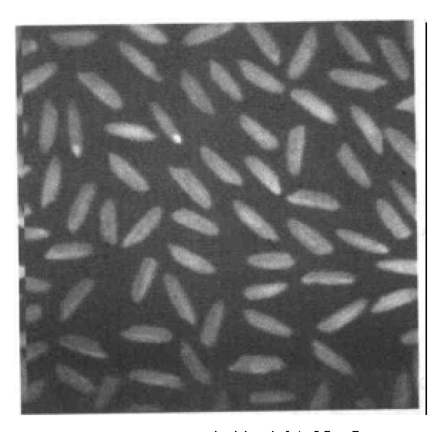


图 4.6 图像的单帧显示

2. 多帧显示

在 MATLAB 中，要同时显示多帧图像阵列，需要调用 montage 函数。其格式具体如下：

```
montage(I)
montage(X,map)
montage(RGB)
h = montage(...)
```

例 4.2.3 图像的多帧显示。

```
A1=imread('rice.tif');
A2=imread('testpat1.tif');
A=cat(2,A1,A2);
montage(A);
```

得到如图 4.7 所示的结果。

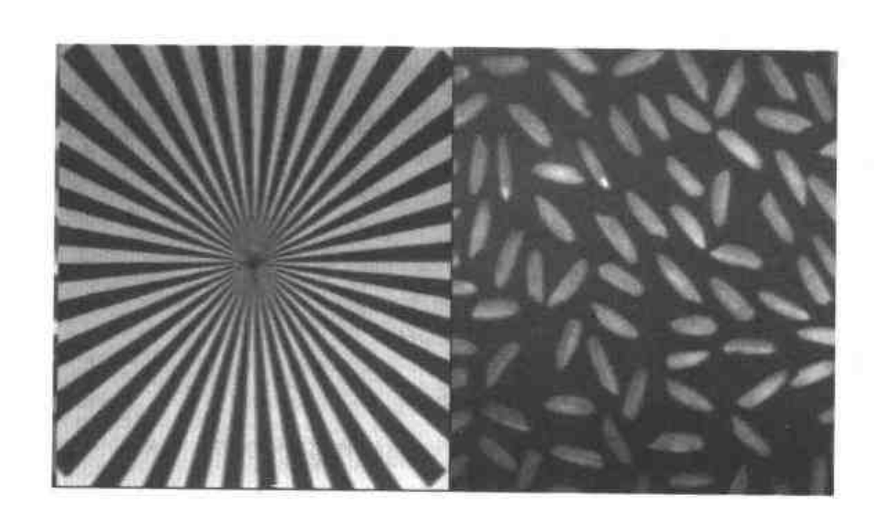


图 4.7 图像的多帧显示

3. 动画显示

利用 immovie 函数，可以从多帧图像阵列中创建 MATLAB 电影动画。其调用格式如下：

mov=immovie(D,map)

需要注意的是，该函数只能用于索引图像。因此，如果用户希望将其它类型的图像阵列转换为电影动画，则必须首先将其类型转换为索引类型，而且函数中所带的参数 D 为一个 m×n×1×k 的 4 维数组，k 代表图像阵列中所包含的帧的数目。

下面的这个例子中，两幅图像都是灰度图像，所以先利用gray2ind函数将它们转换为索引图像，然后还要注意利用已经得到的图像数据矩阵来求数组D。

例 4.2.4 用 immovie 函数创建一个两幅图像交替出现的 MATLAB 电影动画。

```
A1=imread('rice.tif ');
A2=imread('testpat1.tif ');
[B1,map]=gray2ind(A1,256);
[B2,map]=gray2ind(A2,256);          %将两幅灰度图像转换为两幅索引图像
```

```
A=cat(3,B1,B2,B1,B2,B1,B2,B1,B2);      %实现矩阵的合并
s=size(A);
for i=1:s(1,1)                          %该循环用来求代表多帧图像阵列的四维矩阵
    for j=1:s(1,2)
        for k=1:s(1,3)
            D(i,j,1,k)=A(i,j,k);
        end
    end
end
mov=immovie(D,map);
```

4.2.3 图像上的区域缩放

在 MATLAB 中，利用 zoom 命令来实现图像上任一区域的缩放。其调用格式如下：

```
zoom on
zoom off
```

需要说明的是，zoom on 命令使用户可以用鼠标缩放图像。用户可以用鼠标选择一个图像点或拉出一个矩形框，点击左键放大图像，点击右键缩小图像。zoom off 命令关闭缩放功能。

例 4.2.5 用 zoom 函数实现区域的缩放。

```
imshow pic007.jpg;
zoom on;
```

在打开的图像中拖动鼠标左键来选择任意区域，再松开鼠标左键，则原图和放大后的图像分别如图 4.8 和图 4.9 所示。



图 4.8 原图像



图 4.9 放大后的图像（局部）

4.2.4 纹理映射

在 MATLAB 中，纹理映射是一种将二维图像映射到三维图形表面的技术。这种技术通过转换颜色数据使二维图像与三维表面图形保持一致。

MATLAB 的图像处理工具箱中提供了一个专门的函数，即 warp 函数，其作用是将图像作为纹理进行映射，使该图像显示在一个特定的三维空间中。下面具体介绍该函数的用法。

warp 函数的调用格式如下：

```
warp(X,map)
warp(I,n)
warp(RGB)
warp(z,...)
warp(x,y,z,...)
h = warp(...)
```

其中：

warp(X,map)、warp(I,n)、warp(RGB)分别表示将索引图像、灰度图像和真彩图像映射到矩形平面区域上显示。由于矩形平面区域本身就是一个二维图形区域，因而调用这三种格式来显示图像与直接调用 imshow 函数的显示结果是一致的。

warp(z,...)表示将图像映射到曲面 z 上。

warp(x,y,z,...)表示将图像映射到曲面（x,y,z）上。

h = warp(...)表示返回纹理映射后的图形句柄。

还要说明的是，MATLAB 中的纹理映射是利用双线性渐变算法来实现图像的映射的。

例 4.2.6 将一幅真彩图像映射到球面上。

```
RGB=imread('flowers.tif');
[x,y,z]=sphere;                %定义球面
```

```
warp(x,y,z,RGB);
```

映射后的结果如图 4.10 所示。

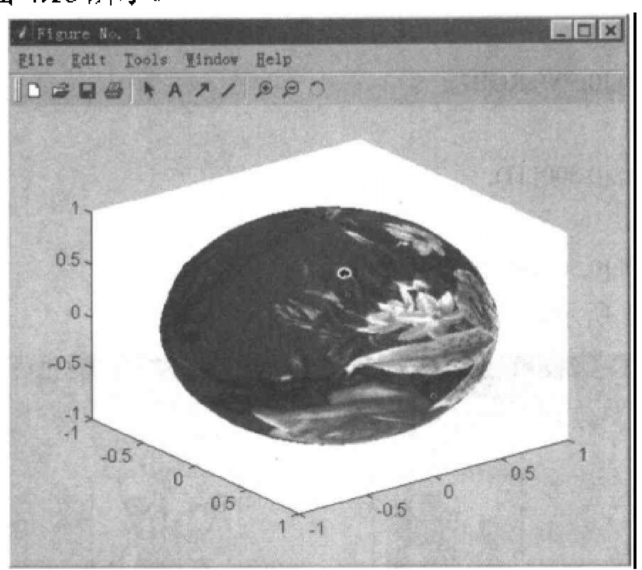


图 4.10　图像的纹理映射

4.2.5 在一个图形窗口中显示多幅图像

为了便于在多幅图像之间进行比较，我们需要将这些要比较的图像显示在一个图形窗口中。MATLAB 的图像处理工具箱就提供了这样一个函数，即 subimage 函数。下面具体介绍这个函数的格式和用法。

subimage 函数的调用格式如下：

```
subimage(X,map)
subimage(I)
subimage(RGB)
subimage(x,y,...)
h = subimage(...)
```

其中：

subimage(X,map)、subimage(I)和 subimage(RGB)分别用于显示索引色、灰度和真彩图像；

subimage(x,y,...)表示将图像按指定的坐标系(x,y)显示，在具体应用时，主要是设置横轴和纵轴的坐标值范围；

h = subimage(...)表示返回图像对象的句柄.

需要说明的是，subimage 函数必须与 subplot 函数一起使用，后者用于指定单个图像的位置，而且 subimage 函数所显示的图像必须是 uint8 或 double 类型。

例 4.2.7　在一个图形窗口中同时显示四幅图像。

```
RGB1=imread('pic005.jpg');
RGB2=imread('pic007.jpg');
I1=imread('rice.tif');
I2=imread('testpat1.tif');
```

```
subplot(2,2,1);    %将图形窗口等分成4部分
subimage([0,500],[0,500],RGB1);
subplot(2,2,2);
subimage([0,500],[0,500],RGB2);
subplot(2,2,3);
subimage([0,500],[0,500],I1);
subplot(2,2,4);
subimage([0,500],[0,500],I2);
```

显示结果如图 4.11 所示。

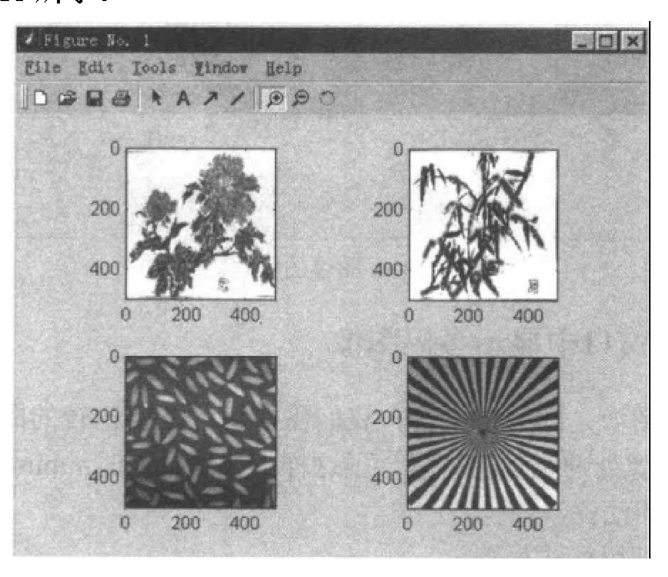


图 4.11　在一个窗口中显示多幅图像

4.3　MATLAB 中的颜色模型

在图像处理和显示的过程中，为了能够正确使用颜色，需要建立颜色模型。颜色模型是三维颜色空间中的一个可见光子集，它包含某个颜色域的所有颜色。常用的颜色模型有 RGB、HSV、NTSC 和 YCbCr。

下面具体介绍这几种模型以及它们之间的相互转换关系。

4.3.1　颜色模型的分类

1. RGB 模型

RGB 模型是面向诸如彩色显示器或打印机之类的硬设备的常见的颜色模型。

该模型基于迪卡尔坐标系统，三个轴分别对应 R（红色）、G（绿色）、B（蓝色）。为了方便，通常将该模型空间归一化为单位立方体，这样所用的 R、G、B 值都落在 0～1 区间。此时，原点对应黑色，离原点最远的立方体的顶点对应白色，而从黑到白的灰度值落在从原点到离原点最远的顶点的连线上。

2. HSV 模型

HSV 模型是面向用户的，是一种复合主观感觉的色彩模型。

HSV 模型对应于圆柱坐标系中的一个圆锥子集，如图 4.12 所示。在 HSV 模型中，H 即色调（hue），表示颜色的种类，取值范围为 0～1，相应的颜色从红、黄、绿、蓝绿、蓝、紫到黑变化，且它的值由绕 V 轴的旋转角决定，每一种颜色和它的补色之间相差 180°；S 即饱和度（saturation），它的取值范围也是 0～1，相应的颜色从未饱和（灰度）向完全饱和（无白色元素）变化；V 即亮度（value），其取值范围同样是 0～1，相应的颜色逐渐变亮。

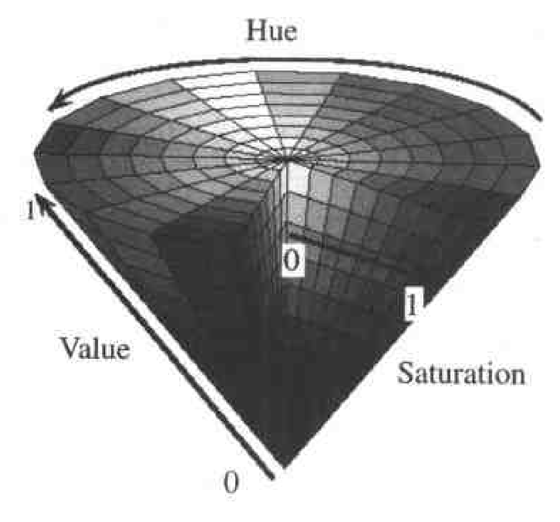


图 4.12 HSV 色彩模型

3. YCbCr 模型

YCbCr 模型是视频图像和数字图像中常用的色彩模型。

在 YCbCr 模型中，Y 为亮度，Cb 和 Cr 共同描述图像的色调，其中 Cb、Cr 分别为蓝色分量和红色分量相对于参考值的坐标。YCbCr 模型中的数据可以是双精度类型，但存储空间为 8 位无符号整型数据空间，且 Y 的取值范围为 16～235，Cb 和 Cr 的取值范围为 16～240。

4. NTSC 模型

NTSC 模型是一种用于电视图像的颜色模型。

NTSC 模型使用的是 Y.I.Q 色彩坐标系，其中，Y 为光亮度，表示灰度信息；I 为色调，Q 为饱和度，均表示颜色信息。因此，该模型的主要优点就是将灰度信息和颜色信息区分开来。

Y.I.Q 坐标系与 R.G.B 坐标系的转换关系为

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.956 & 0.621 \\ 1.000 & -0.272 & -0.647 \\ 1.000 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

以上简单地介绍了各种颜色模型，下面介绍它们之间的相互转换关系。

4.3.2 颜色模型的转换

表 4.1 给出了 MATLAB 的图像处理工具箱中用于色彩模型转换的函数。

表 4.1　用于颜色模型转换的函数

函数名	函数功能
hsv2rgb	HSV 模型转换为 RGB 模型
ntsc2rgb	NTSC 模型转换为 RGB 模型
rgb2hsv	RGB 模型转换为 HSV 模型
rgb2ntsc	RGB 模型转换为 NTSC 模型
rgb2ycbcr	RGB 模型转换为 YCbCr 模型
ycbcr2rgb	YCbCr 模型转换为 RGB 模型

下面介绍这些函数的用法。

1. rgb2hsv 函数

该函数用来将 RGB 模型转换为 HSV 模型。其调用格式如下：

hsvmap = rgb2hsv(rgbmap)

HSV = rgb2hsv(RGB)

其中：

hsvmap = rgb2hsv(rgbmap)表示将 RGB 空间中 $m \times 3$ 的色彩表 rgbmap 转换成 HSV 色彩空间的颜色映射表 hsvmap;

HSV = rgb2hsv(RGB)表示将真彩图像 RGB 转换为 HSV 色彩空间中的图像 HSV。

例 4.3.1　把一幅真彩图像转换为一个 HSV 模型空间对应的图像。

```
RGB=imread('hr.jpg');
HSV=rgb2hsv(RGB);
subplot(1,2,1);
subimage(RGB);
title('原图像');
subplot(1,2,2);
subimage(HSV);
title('变换后的图像');
```

显示的结果如图 4.13 所示。

2. rgb2ntsc 函数

该函数用来将 RGB 模型转换为 NTSC 模型。其调用格式如下：

yiqmap = rgb2ntsc(rgbmap)

YIQ = rgb2ntsc(RGB)

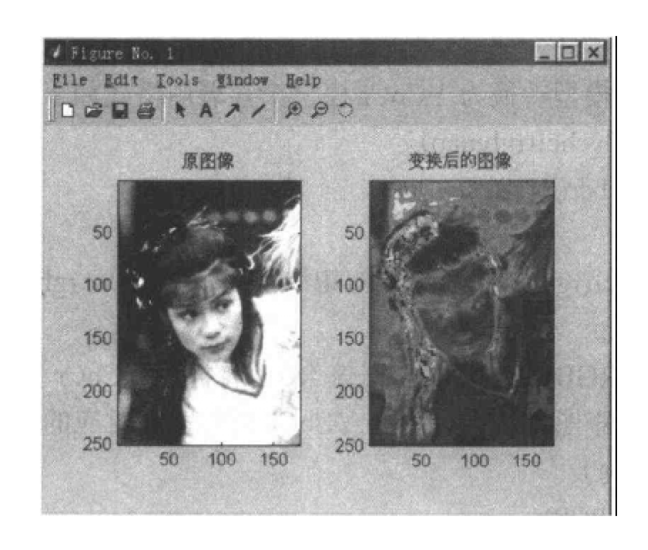


图 4.13　RGB 图像与 HSV 图像的比较

其中：

yiqmap = rgb2ntsc(rgbmap)表示将 RGB 空间中 m × 3 的色彩表 rgbmap 转换成 YIQ 空间中的颜色映射表 yiqmap；

YIQ = rgb2ntsc(RGB)将真彩图像 RGB 转换为 YIQ 色彩空间中的图像 YIQ。

例 4.3.2　把一幅真彩图像转换为一个 NTSC 模型空间对应的图像。

```
RGB=imread('hr.jpg');
YIQ=rgb2ntsc(RGB);
I=YIQ(:,:,1);
subplot(1,2,1);
subimage(RGB);
title('原图像');
subplot(1,2,2);
subimage(I);
title('变换后的图像');
```

执行的结果如图 4.14 所示。

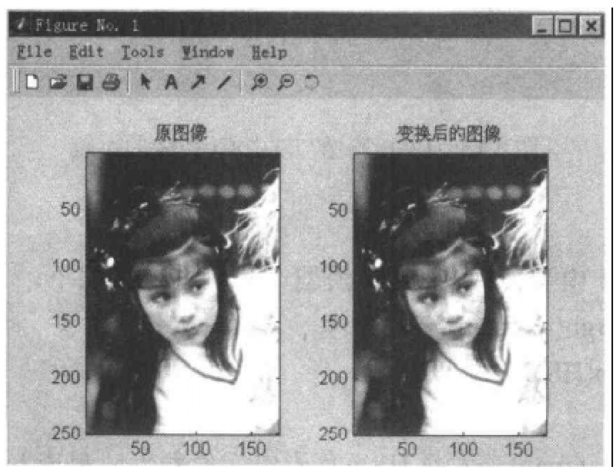


图 4.14　RGB 图像与 NTSC 图像的比较

3. rgb2ycbcr 函数

该函数用来将 RGB 模型转换为 YCbCr 模型。其调用格式如下：

ycbcrmap = rgb2ycbcr(rgbmap)

YCbCr = rgb2ycbcr(RGB)

其中：

ycbcrmap = rgb2ycbcr(rgbmap)表示将 RGB 空间中的色彩表 rgbmap 转换为 YCbCr 空间中的颜色映射表 ycbcrmap;

YCbCr = rgb2ycbcr(RGB)表示将真彩图像 RGB 转换为 YCbCr 空间中的图像 YCbCr。

例 4.3.3 把一幅真彩图像转换为一个 YCbCr 模型空间对应的图像。

```
RGB=imread('hr.jpg');
YCbCr=rgb2ycbcr(RGB);
subplot(1,2,1);
subimage(RGB);
title('原图像');
subplot(1,2,2);
subplot(1,2,2);
subimage(YCbCr);
title('变换后的图像');
```

执行的结果如图 4.15 所示。



图 4.15 RGB 图像与 YCbCr 图像的比较

4. hsv2rgb 函数

该函数用来将 HSV 模型转换为 RGB 模型。其调用格式如下：

rgbmap = hsv2rgb(hsvmap)

RGB = hsv2rgb(HSV)

其中：

rgbmap = hsv2rgb(hsvmap)表示将 HSV 色彩空间的颜色映射表 hsvmap 转换成 RGB 空间

中的色彩表 rgbmap，rgbmap 和 hsvmap 都是 m × 3 的矩阵；

RGB = hsv2rgb(HSV)表示将 HSV 色彩空间的图像 HSV 转换为真彩图像 RGB。

5. ntsc2rgb 函数

该函数用来将 NTSC 模型转换为 RGB 模型。其调用格式如下：

rgbmap = ntsc2rgb(yiqmap)

RGB = ntsc2rgb(YIQ)

其中：

rgbmap = ntsc2rgb(yiqmap)表示将 YIQ 空间中 m × 3 的颜色映射表 yiqmap 转换成 RGB 空间中的色彩表 rgbmap；

RGB = ntsc2rgb(YIQ) 表示将 YIQ 色彩空间的图像 YIQ 转换为真彩图像 RGB。

6. ycbcr2rgb 函数

该函数用来将 YCbCr 模型转换为 RGB 模型。其调用格式如下：

rgbmap = ycbcr2rgb(ycbcrmap)

RGB = ycbcr2rgb(YCBCR)

其中：

rgbmap = ycbcr2rgb(ycbcrmap)表示将 YCbCr 空间中的颜色映射表 ycbcrmap 转换为 RGB 空间中的色彩表 rgbmap。

RGB = ycbcr2rgb(YCBCR)表示将 YCbCr 空间中的图像 YCBCR 转换为真彩图像 RGB。

图形编程基础篇

第五章　图像的几何操作及基于区域的处理

图像的几何操作包括图像的缩放、旋转和剪切。

我们在图像处理的过程中，有时并不需要对整个图像进行处理，而只要对图像中的某个特定区域进行处理。

本章将对图像的几种几何操作和对图像的特定区域的处理进行较为详细的介绍，使读者能够了解相关的原理，并且运用 MATLAB 来实现上述的操作。

本章主要内容：

★ 图像插值的基本原理
★ 图像的插值缩放
★ 图像的插值旋转
★ 图像的剪切
★ 基于区域的图像处理

5.1　图像插值的基本原理

在对图像的几种几何操作中，图像的缩放和旋转都要用到插值操作。插值算法的好坏直接关系到图像的失真程度，插值函数的设计是插值算法的核心问题。因此，在具体介绍这几种几何操作前，有必要先介绍插值的基本原理。

插值通常是利用曲线拟合的方法，通过离散的采样点建立一个连续函数，用这个重建的函数便可以求出任意位置的函数值，如图 5.1 所示。

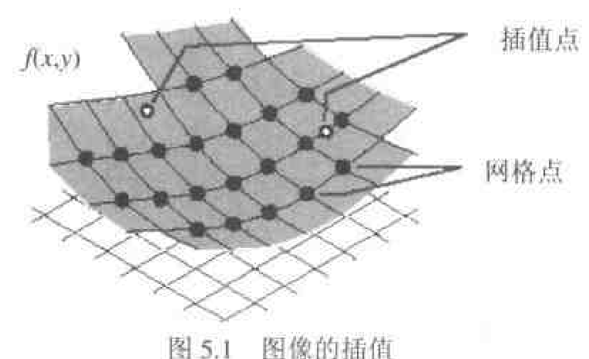


图 5.1　图像的插值

对于等间隔离散数据，插值可以表示为

$$f(x)=\sum_{k=0}^{K-1}C_k h(x-x_k)$$

其中，h 为插值核；C_k 为权系数。

插值算法的数值精度及计算开销直接与插值核有关，插值核的设计是插值算法的核心。MATLAB 的图像处理工具箱提供了三种插值方法：

- 最近邻插值（Nearest neighbor interpolation）；
- 双线性插值（Bilinear interpolation）；
- 双三次插值（Bicubic interpolation）。

下面具体介绍这几种插值方法的原理和特点。

5.1.1　最近邻插值

从计算量的角度来说，最近邻插值是最简单的插值。在这种算法中，每一个插值输出像素的值就是在输入图像中与其最临近的采样点的值。算法的数学表示为

$$f(x)=f(x_k)\;,\quad \frac{1}{2}(x_{k-1}+x_k)<x\leqslant\frac{1}{2}(x_k+x_{k+1})$$

最近邻法的插值核及其傅立叶谱如图 5.2 所示。

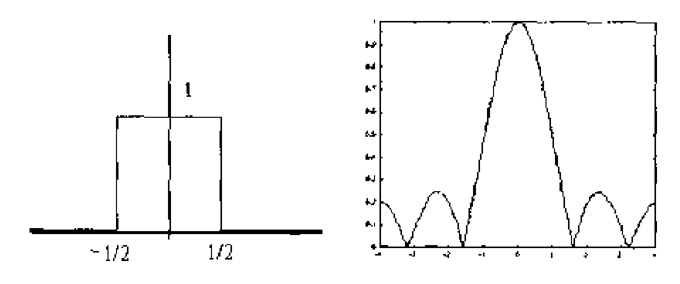


图 5.2　最近邻法的插值核及其傅立叶谱

最近邻插值是工具箱函数缺省使用的插值方法。对于索引图像来讲，它也是惟一可行的方法。不过，最近邻法插值核的频域特性并不好，从它的傅立叶谱上可以看出，它与理想低通滤波器的性质相差较大。当用这种方法实现大倍数放大处理时，在图像中可以明显地看出块状效应。

5.1.2　双线性插值

在双线性插值中，输出像素值是它在输入图像中 2×2 邻域采样点的平均值。这样看来，它相当于一种平滑操作。

双线性插值的插值核为三角函数，其函数图形及傅立叶谱如图 5.3 所示。

从其傅立叶谱上可以看出双线性插值函数的频域特性优于最近邻插值函数，其频谱的旁瓣远小于主瓣，表明它的带阻特性较好。不过，仍有大量高频成分漏入通频带，造成了一定的混叠。此外，通频带在一定程度上被减弱，由此可使插值后的图像变模糊，从而损

失一些细节。

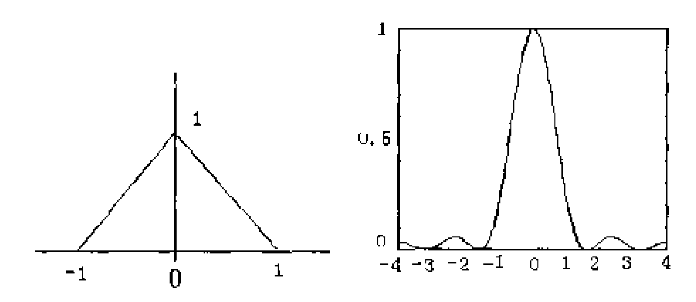


图 5.3 双线性插值的插值核及其傅立叶谱

5.1.3 双三次插值

双三次插值的插值核为三次函数，其插值邻域的大小为4×4。它的插值效果比较好，但相应的计算量也较大，一般很少使用。

双三次插值法的插值核及其傅立叶谱如图 5.4 所示。

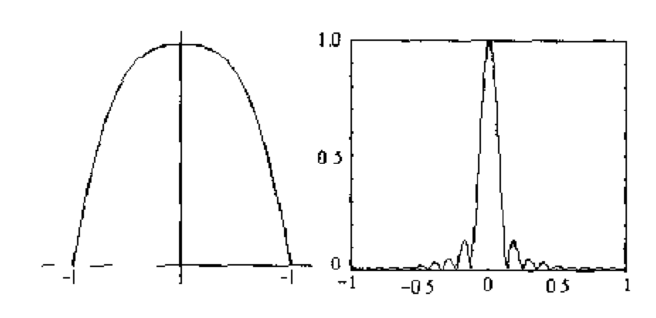


图 5.4 双三次插值的插值核及其傅立叶谱

5.2 图像的插值缩放

在 MATLAB 中，用函数 imresize 来实现对图像的放大或缩小。插值方法可选用以上三种方法中的任意一种，缺省的插值方法是最近邻法。

该函数的调用格式如下：

```
B = imresize(A,m,method)
B = imresize(A,[mrows ncols],method)
B = imresize(...,method,n)
B = imresize(...,method,h)
```

其中：

参数 method 用于指定插值的方法，可选的值为'nearest'（最近邻法）、'bilinear'（双线性插值）、'bicubic'（双三次插值），缺省值为'nearest'；

B = imresize(A,m,method)表示返回原图 A 的 m 倍放大图像(m 小于 1 时实际上是缩小)；

B = imresize(A,[mrows ncols],method)表示返回一个 mrows 行、ncols 列的图像，若与原图的长/宽比不同，则图像会产生变形；

在使用'bilin'和'bicubic'方法缩小图像时，为消除引入的高频成分，imresize 使用了一个前端平滑滤波器，缺省的滤波器尺寸为11×11；用户也可通过参数 n 指定滤波器的尺寸，即 B = imresize(...,method,n);

B = imresize(...,method,h)表示使用用户设计的插值核 h 进行插值，h 可以看作一个二维 FIR 滤波器。

例 5.2.1 使用不同插值方法放大图像。

```
I=imread('rice.tif');
imshow(I);
I1=imresize(I,1.5,'nearest neighber');  %最近邻插值
figure,imshow(I1),
I2=imresize(I,1.5,'bilibear');          %双线性插值
figure,imshow(I2);
I3=imresize(I,1.5,'bicubic');           %双三次插值
figure,imshow(I3);
```

图 5.5 为原图像，图 5.6、图 5.7 和图 5.8 为采用了上述三种不同的方法放大图像后的结果。

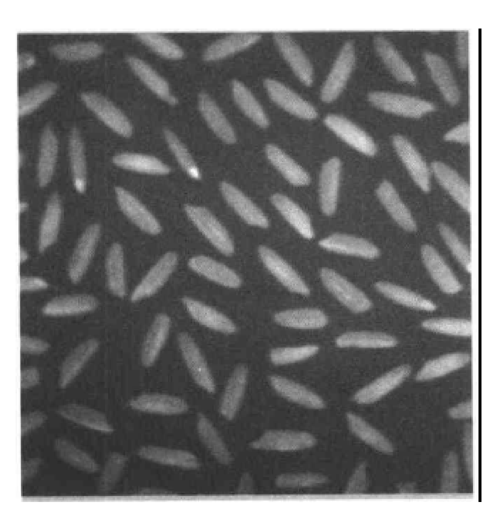


图 5.5 原图像

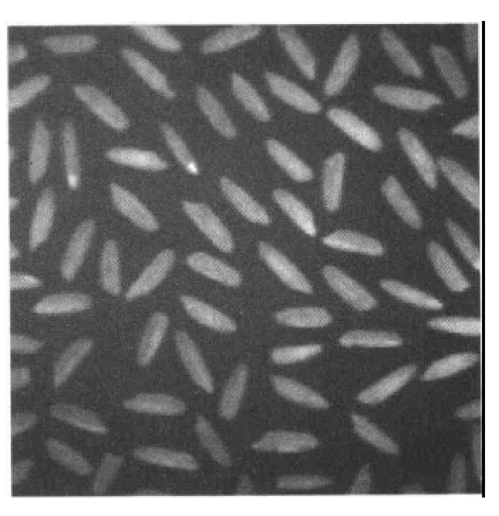


图 5.6 近邻插值法放大 1.5 倍图像的结果

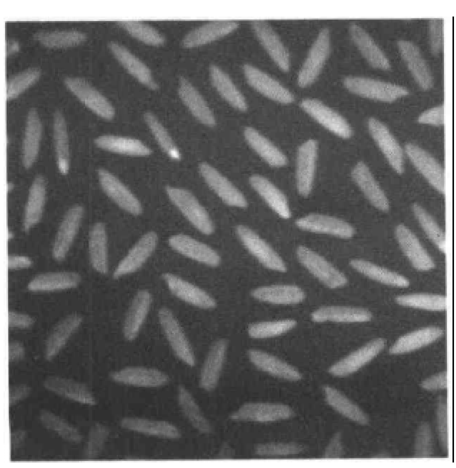


图 5.7 双线性插值法放大 1.5 倍图像的结果

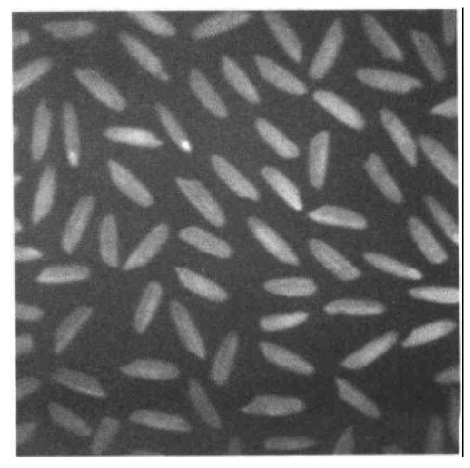


图 5.8 双三次插值法放大 1.5 倍图像的结果

由以上各图可看出，最近邻插值的效果是最差的，双三次插值是最好的，双线性插值的效果介于两者之间。

5.3 图像的插值旋转

在 MATLAB 中，用函数 imrotate 来实现对图像的插值旋转。插值方法也可选用三种方法中的任意一种，缺省的插值方法是最近邻法。

该函数的调用格式如下：

```
B = imrotate(A,angle,method)
B = imrotate(A,angle,method,'crop')
```

其中，参数 method 用于指定插值的方法，可选的值为'nearest'（最近邻法）、'bilinear'（双线性插值）、'bicubic'（双三次插值），缺省值为'nearest'。参数 angle 代表旋转的角度。

一般来说，旋转后的图像会比原图大，用户可以指定'crop'参数对旋转后的图像进行剪切（取图像的中间部分），使返回的图像与原图大小相同。

例 5.3.1 图像的插值旋转。

```
I=imread('eight.tif');
I1=imrotate(I,30,'bilinear','crop');
I2=imrotate(I,30,'nearest neighber');
imshow(I)
figure,imshow(I1)
figure,imshow(I2)
```

I1、I2 都将图像 eight.tif 旋转了 30°、I2 采用的是最近邻插值，且旋转后没有剪裁图像，而 I1 则是采用双线性插值，且旋转后将图像剪裁成与原图像大小相等的图像。

原图以及I1和I2的显示结果分别如图5.9、图5.10及图5.11所示。

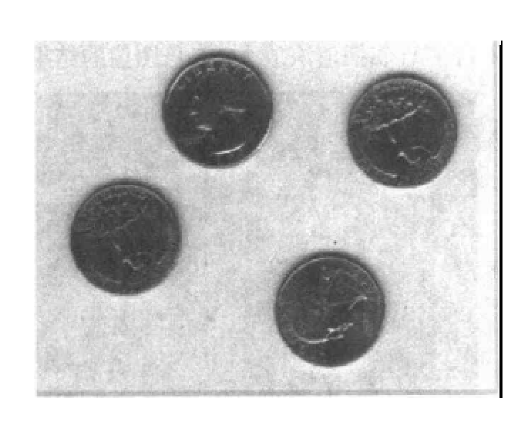


图 5.9 原图像

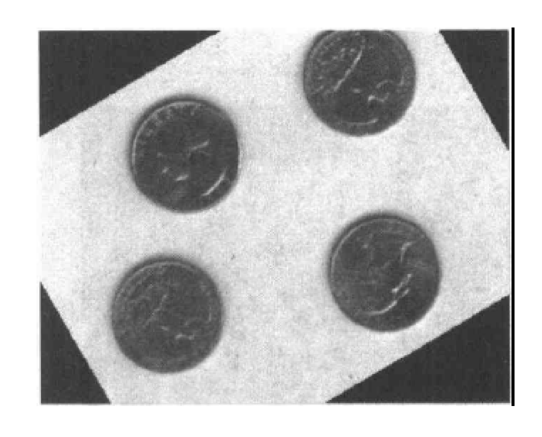


图 5.10 旋转 30° 后的图像（带'crop'参数）

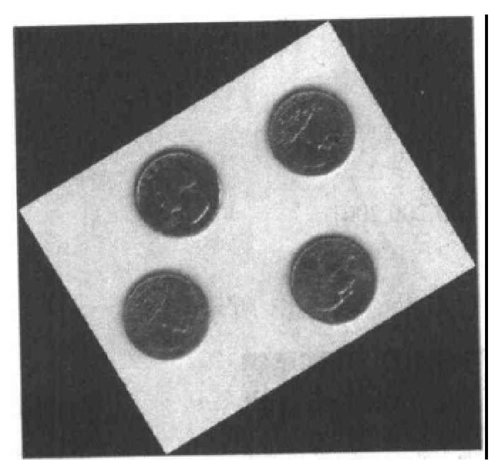


图 5.11　旋转 30° 后的图像（不带'crop'参数）

5.4　图像的剪切

在 MATLAB 中，用函数 imcrop 实现对图像的剪切操作。该操作剪切的是图像中的一个矩形子图，用户可以通过参数指定这个矩形四个顶点的坐标，也可以交互地用鼠标选取这个矩形。

imcrop函数的调用格式如下：

```
I2 = imcrop(I)
X2 = imcrop(X,map)
RGB2 = imcrop(RGB)

I2 = imcrop(I,rect)
X2 = imcrop(X,map,rect)
RGB2 = imcrop(RGB,rect)

[...] = imcrop(x,y,...)
[A,rect] = imcrop(...)
[x,y,A,rect] = imcrop(...)
```

其中：

I2 = imcrop(I)、X2 = imcrop(X,map)和 RGB2 = imcrop(RGB) 分别表示交互式地对灰度图像、索引图像和真彩图像进行剪切操作；

I2 = imcrop(I,rect) 、X2 = imcrop(X,map,rect)和 RGB2 = imcrop(RGB,rect)分别表示按指定的矩形框 rect 剪切图像，rect 是一个 4 元向量[xmin ymin width height]；

[...] = imcrop(x,y,...)表示在指定坐标系(x,y)中剪切图像；

[A,rect] = imcrop(...)和[x,y,A,rect] = imcrop(...)表示在剪切图像的同时返回剪切框的参数 rect。

例 5.4.1 图像 flower.tif 中剪切了一块子图，顶点坐标为(40，50)、(40，250)、(240，50）和（240，250)。

```
RGB=imread('flowers.tif ');
imshow(RGB)
RGB2=imcrop(RGB,[40,50,200,200]);
figure,imshow(RGB2)
```

原图见图 5.12，剪切后的结果如图 5.13 所示。



图 5.12 原图像



图 5.13 剪切后的图像

5.5 基于区域的图像处理

要对一幅图像的特定的区域进行处理时，首先要定义这个我们感兴趣的区域。在 MATLAB 中，对这个特定区域的定义是通过创建一个二进制的 mask（为一幅二进制图像）来实现的，暂且称其为 mask 图像。该 mask 图像与原图像具有相同的尺寸，我们所选定的区域的像素在 mask 中的值为 1（即为白色），其余部分的像素值为 0（即为黑色）。这样，就通过 mask 图像实现了对特定区域的选择。

下面具体介绍如何创建一个 mask 图像来实现特定区域的选择。

5.5.1 多边形选择法

在 MATLAB 中，用函数 roipoly 可以设定一个多边形区域。其调用格式如下：

```
BW = roipoly(I,c,r)
BW = roipoly(I)
BW = roipoly(x,y,I,xi,yi)
[BW,xi,yi] = roipoly(...)
[x,y,BW,xi,yi] = roipoly(...)
```

其中：

roipoly 函数返回二值图像 BW，选中的区域为白色，其余的部分为黑色；

BW = roipoly(I,c,r)表示用向量 c、r 来指定多边形各顶点的 X、Y 轴的坐标；

BW = roipoly(I)表示让用户交互地选择多边形区域，用鼠标左键选择顶点，用空格键或

Delete 键撤消选择，回车键确认；

BW = roipoly(x,y,I,xi,yi)表示在指定的坐标系 X.Y 下选择由向量 xi、yi 指定的多边形区域；

[BW,xi,yi] = roipoly(...)表示交互地选择多边形区域，并返回多边形各顶点的坐标；

[x,y,BW,xi,yi] = roipoly(...)表示在交互地选择多边形区域后，返回多边形的各顶点在指定的坐标系 X.Y 下的坐标。

例 5.5.1 选择图像中的六边形区域。

```
I = imread('eight.tif');
c = [222 272 300 270 221 194];        %定义多边形顶点的 X 坐标
r = [21 21 75 121 121 75];            %定义多边形顶点的 Y 坐标
BW = roipoly(I,c,r);
imshow(I);
figure;
imshow(BW);
```

原图像和 mask 图像分别如图 5.14 和图 5.15 所示。

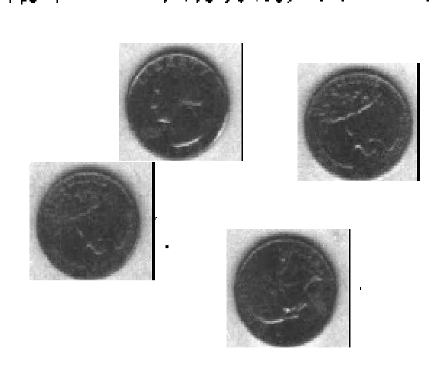


图 5.14 原图像

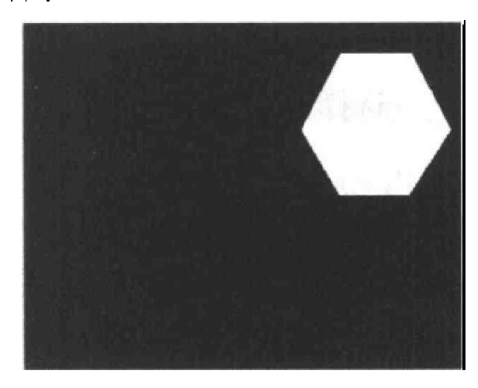


图 5.15 mask 图像

5.5.2 灰度选择法

在 MATLAB 中，用户可以用 roicolor（）函数来根据颜色和灰度范围设定感兴趣的区域。

roicolor（）函数的调用格式如下：

BW = roicolor(A,low,high)

BW = roicolor(A,v)

其中：

BW = roicolor(A,low,high)表示按指定的灰度范围分割图像，返回代表 mask 图像的数据矩阵 BW，[low high]为所要选择区域的灰度范围；

BW = roicolor(A,v)表示按向量 v 中指定的灰度值来选择区域。

例 5.5.2 选中灰度值在 150～200 之间的图像中的区域。

```
I = imread('rice.tif');
```

```
BW = roicolor(I,150,200);
imshow(I);
figure;
imshow(BW)
```

原图及选择后的结果如图 5.16 和 5.17 所示。

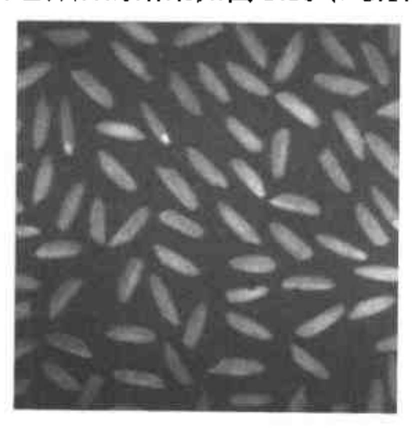


图 5.16 原图像



图 5.17 按灰度选择后的区域

5.5.3 其它选择方法

除了使用上述两种方法生成 mask 图像外，用户还可以选用任意的二值图像来作为 mask 图像，但该二值图像必须具有与原始图像相同的尺寸。

例 5.5.3 想对一个数据矩阵为 I 的灰度图像进行过滤，过滤时只滤去灰度值大于 200 的像素，则我们可以用下面的命令生成一个合适的 mask。

```
I=imread('rice.tif ');
BW=(I>200);
```

5.5.4 对指定区域的滤波

在 MATLAB 中，用函数 roifilt2 实现对指定区域的滤波。其调用格式如下：

```
J = roifilt2(h,I,BW)
J = roifilt2(I,BW,fun)
J = roifilt2(I,BW,fun,P1,P2,...)
```

其中：

J = roifilt2(h,I,BW)表示使用滤波器 h 对图像 I 中用 mask 选中的区域进行滤波；

J = roifilt2(I,BW,fun) 和 J = roifilt2(I,BW,fun,P1,P2,...) 表示对图像 I 中用 mask 选中的区域作函数运算 fun，其中 fun 是描述函数运算的字符串，参数为 P1,P2,...，返回图像 J 在选中区域的像素为 I 经 fun 运算的结果，其余部分的像素值为 I 的原始值。

例 5.5.4 对指定区域进行锐化滤波。

```
I = imread('eight.tif');
c = [222 272 300 270 221 194];
```

```
r = [21 21 75 121 121 75];
BW = roipoly(I,c,r);
h = fspecial('unsharp');
J = roifilt2(h,I,BW);
subplot(1,2,1),imshow(I);
subplot(1,2,2),imshow(J);
```

结果如图 5.18 所示。

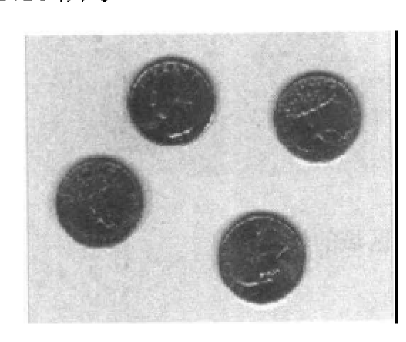
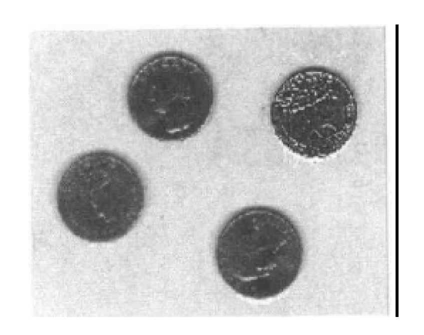



图 5.18　对指定区域的锐化滤波

5.5.5　对指定区域的填充

在 MATLAB 中，用函数 roifill 实现对指定区域的填充。其调用格式如下：

```
J = roifill(I,c,r)
J = roifill(I)
J = roifill(I,BW)
[J,BW] = roifill(...)
J = roifill(x,y,I,xi,yi)
[x,y,J,BW,xi,yi] = roifill(...)
```

其中：

J = roifill(I,c,r)表示填充由向量 c、r 指定的多边形，c 和 r 分别为多边形各顶点的 X、Y 轴的坐标；

J = roifill(I)表示由用户交互式地选取填充的区域。用鼠标左键选择多边形的各顶点，回车键表示结束，空格键或 Delete 键表示取消一个选择；

J = roifill(I,BW)表示填充由 BW 决定的 mask 图像所选择的区域；

[J,BW] = roifill(...)表示在填充区域的同时还返回代表 mask 图像的数据矩阵 BW；

J = roifill(x,y,I,xi,yi)和[x,y,J,BW,xi,yi] = roifill(...)表示在指定的坐标系 X.Y 下填充由向量 xi、yi 指定的多边形区域。

例 5.5.5　填充一个指定的区域。

```
I = imread('eight.tif');
c = [222 272 300 270 221 194];
r = [21 21 75 121 121 75];
```

```
J = roifill(I,c,r);
subplot(1,2,1),imshow(I)
subplot(1,2,2),imshow(J)
```

填充的结果如图 5.19 所示。

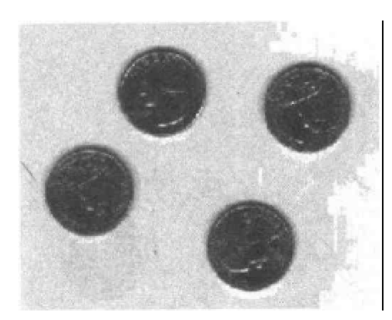
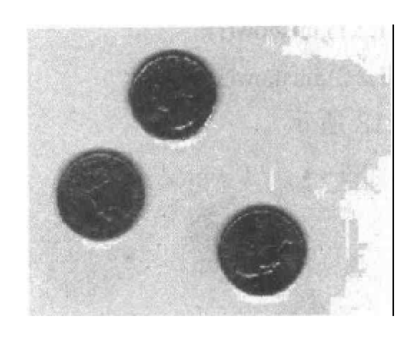



图 5.19 对指定区域的填充

图像处理技术篇

第六章 图像变换

为了快速有效地对图像进行处理和分析，如进行图像增强、图像分析、图像复原、图像压缩等，常常需要将原定义在图像空间的图像以某种形式转换到另外一些空间，并利用在这些空间的特有性质方便地进行一定的加工，最后再转换回图像空间，以得到所需的效果。因此，图像变换是许多图像处理和分析的基础。

图像变换的种类很多，在 MATLAB 中主要是：快速傅立叶变换、离散余弦变换和 Radon 变换。

本章主要内容：

- ★ 傅立叶变换
- ★ 离散余弦变换
- ★ Radon 变换

6.1 傅立叶变换

傅立叶变换是对线性系统进行分析的一个有力工具，它将图像从空域变换到频域，使我们能够定量地分析诸如数字化系统、采样点、电子放大器、卷积滤波器、噪声、显示点等的作用(效应)。把傅立叶变换的理论同其物理解释相结合，将有助于解决大多数图像处理问题。

在数字图像处理中，输入图像和输出图像通常都是二维的，一般表示成二维数字矩阵，因此，这里直接讨论二维傅立叶变换、二维 DFT、二维 FFT。

6.1.1 二维连续傅立叶变换

二维傅立叶变换的定义如下：

$$F(u,v)=\sum_{x=-\infty}^{\infty}\sum_{y=-\infty}^{\infty}f(m,n)e^{-jux}e^{-jvy}$$

其中，u 和 v 是频率变量，其单位是弧度/采样单位；F(u,v)通常称为 f(x,y) 的频域表示。F(u,v)是复值函数，在 u 和 v 上都是周期性的，且周期为 2π。因为其具有周期性，通常只

显示 $-\pi \leqslant u, v \leqslant \pi$ 的范围。注意 F(0, 0)是f(x,y)的所有值之和，因此，F(0, 0)通常称为傅立叶变换的恒定分量或DC分量(DC表示直流，是电子工程学术语，指电压恒定的电源，与电压值呈正弦变化的电源不同)。如果f(x,y)是一幅图像，则F(u,v)是它的谱，变量u对应着x轴，v对应着y轴。

二维傅立叶反变换定义如下：

$$f(x,y)=\frac{1}{2\pi}\int_{u=-\pi}^{\pi}\int_{v=-\pi}^{\pi}F(u,v)e^{jux}e^{jvy}dudv$$

粗略地讲，此方程表明 f(x,y)可以表示为无穷多个不同频率的复指数幂(正弦的)之和，在频率(u,v)上贡献的幅值和相位由F(u,v)给出。

同时定义二维傅立叶变换的频谱、相位谱和功率谱(频谱密度)如下：

$$|F(u,v)|=\left[R^2(u,v)+I^2(u,v)\right]^{\frac{1}{2}}$$

$$\Phi(u,v)=\arctan\left[I(u,v)/R(u,v)\right]$$

$$P(u,v)=|F(u,v)|^2=R^2(u,v)+I^2(u,v)$$

其中R(u,v)和I(u,v)分别为F(u,v)的实部和虚部。

由于幅度谱中表现了一些可辨认的结构，而相位谱看起来是完全随机的，因此一般可能认为幅度谱比较重要，但事实却恰恰相反。如果忽略相位信息，(将相位设为零)然后进行反变换，从得到的图6.4中完全看不出来与原图有什么相似之处；如果忽略幅值信息(在进行反变换之前将幅值设为常数)，从得到的图6.5中，还可以辨认出肖像的轮廓，这说明相位谱在数字图像处理中有着更重要的潜在作用，在处理图像的过程中一定不能忽略相位谱的作用。

幅值谱表明了各正弦分量出现的多少，而相位信息表明了各正弦分量在图像中出现的位置，图6.1～图6.5表明打乱了各正弦分量出现的位置将造成毁灭性的影响。但是只要各正弦分量保持原位，则对于图像整体来说，幅值就显得不那么重要。出于这些原因，大多数实用滤波器都只影响幅值，而几乎不改变相位信息。



图6.1　原始图像

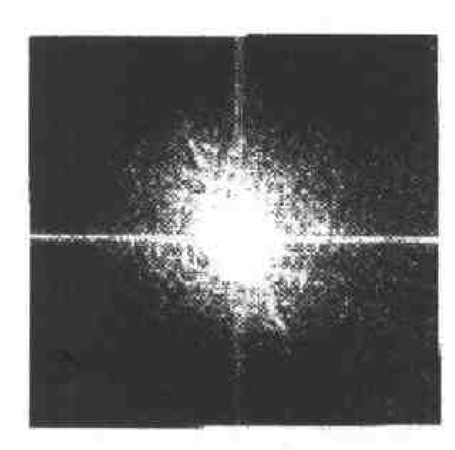


图6.2　幅值谱

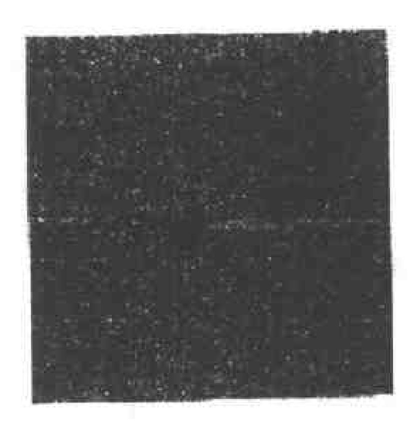


图 6.3 相位谱

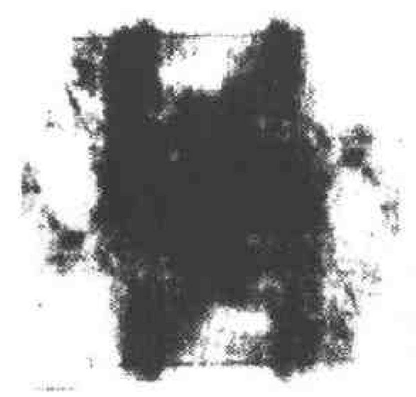


图 6.4 由幅值谱重构的图像



图 6.5 出相位谱重构的图像

二维傅立叶变换有很多重要性质，例如可分离性、平移性、周期性、共轭对称性、旋转性、满足分配律、可进行尺度变换(缩放)、卷积等等，这些性质使得在频域中对图像进行操作变得很简单，从而简化了整个处理过程，因此在图像处理中都有重要应用。表 6.1 列出了其中的一些性质。

表 6.1 傅立叶变换的性质

性质	空 域	频 域
加法定理		$F(u,v)+G(u,v)$
相似性定理	$f(ax,by)$	$\frac{1}{\lvert ab\rvert}F(\frac{u}{a},\frac{v}{b})$
位移定理	$f(x-a,y-b)$	$e^{-j2\pi(au+bv)}F(u,v)$
卷积定理	$f(x,y)*g(x,y)$	$F(u,v)G(u,v)$
可分离乘积	$f(x)g(y)$	$F(u)G(v)$
微分	$(\frac{\partial}{\partial x})^m(\frac{\partial}{\partial y})^n f(x,y)$	$(j2\pi u)^m(j2\pi v)^n F(u,v)$
旋转	$f(x\cos\theta+y\sin\theta,\ -x\sin\theta+y\cos\theta)$	$F(u\cos\theta+v\sin\theta,\ -u\sin\theta+v\cos\theta)$
拉普拉斯	$\nabla^2 f(x,y)=\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}f(x,y)$	$-4\pi^2(u^2+v^2)F(u,v)$
Rayleigh 定理	$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\lvert f(x,y)\rvert^2 dxdy=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\lvert F(u,v)\rvert^2 dudv$	

1. 平移性

由位移定理可以看出，$f(x,y)$ 的平移不影响其傅立叶变换的幅值，不管图像沿 x 轴平移，沿 y 轴平移，还是在两个方向上都有平移，对应的幅值谱都和原始图像是完全一样的(见

图 6.6～图 6.11)。根据此性质，当对图像进行傅立叶变换后，如果原点不在图像的中心，则可以先进行平移，对平移后的幅值谱进行处理，然后再将处理后的幅值谱平移回原位，这样操作不但对得到的结果没有影响，还省掉了幅值谱的原点不在图的中心带来的麻烦。

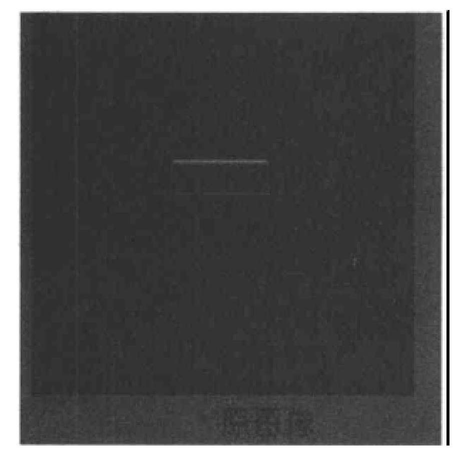


图 6.6　原始图像

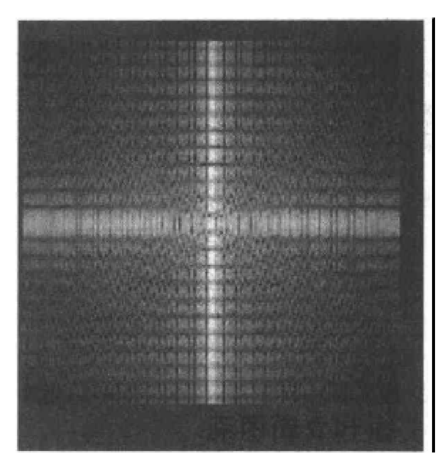


图 6.7　原图的傅立叶谱

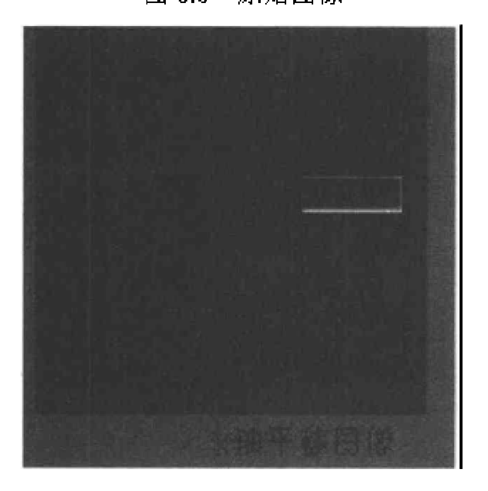


图 6.8　x 轴平移图像

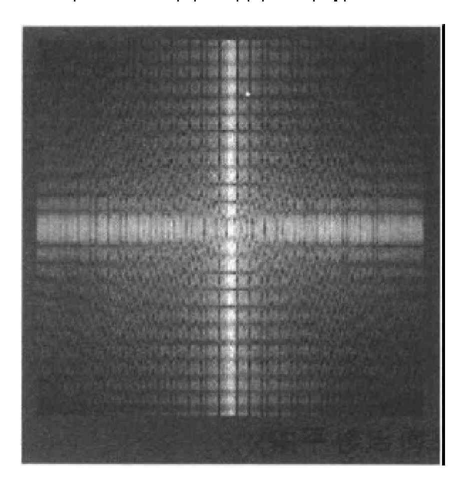


图 6.9　x 轴平移图像的傅立叶谱

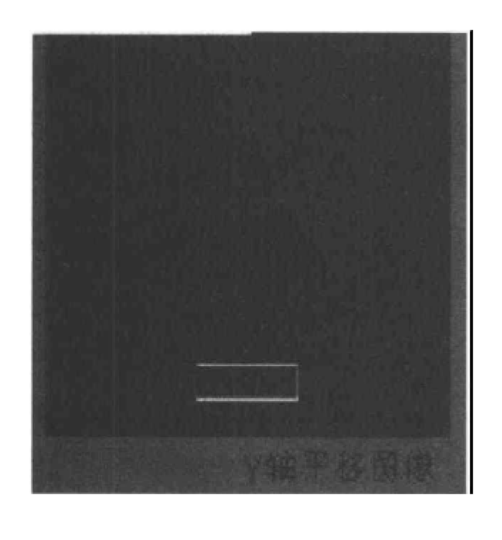


图 6.10　y 轴平移图像

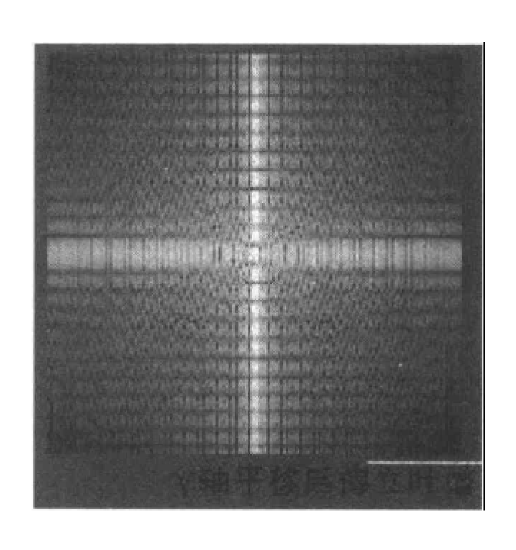


图 6.11　y 轴平移图像的傅立叶谱

2. 旋转性质

由列表可知，f(x,y)旋转θ角，则其傅立叶变换F(u,v)也旋转θ角；F(u,v)旋转θ角，则f(x,y)也旋转θ角。如图6.12～图6.15所示，将原始图像旋转了45°，而相应的傅立叶谱也旋转了45°。

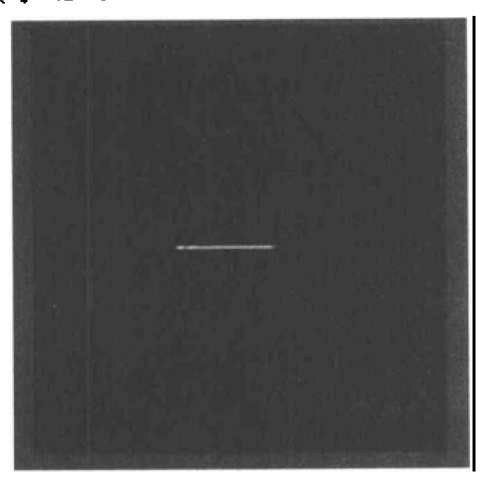


图6.12　原始图像

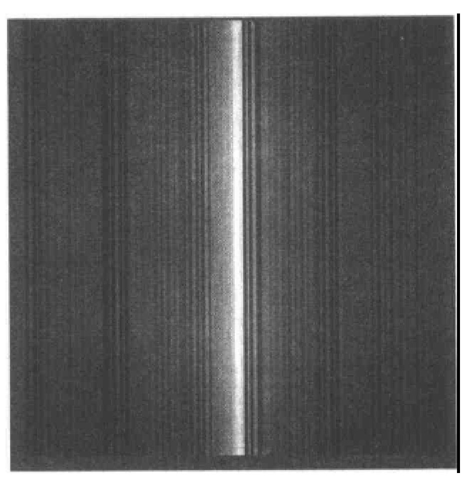


图6.13　原图的傅立叶谱

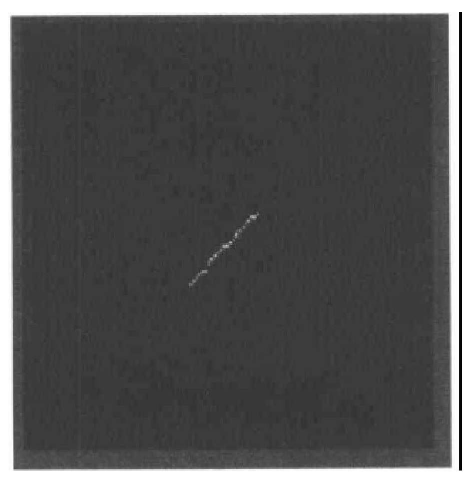


图6.14　旋转后的图像

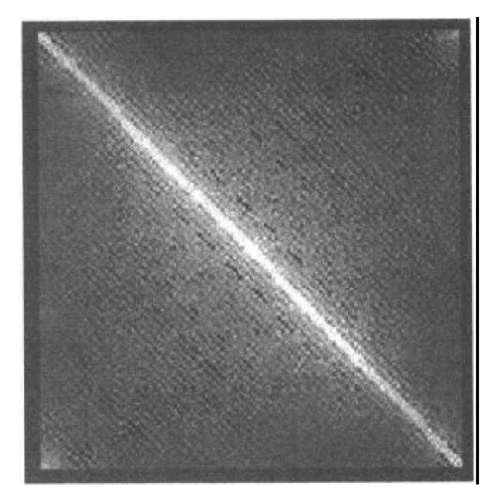


图6.15　旋转图像的傅立叶谱

6.1.2　二维离散傅立叶变换(DFT)

二维M×N的DFT变换和逆DFT变换分别定义如下：

$$F(m,n)=\sum_{i=0}^{M-1}\sum_{k=0}^{N-1}f(i,k)e^{-j(2\pi/M)mi}e^{-j(2\pi/N)nk}$$

其中，m=0，1，…，M-1；n=0，1，…，N-1。

$$f(i,k)=\frac{1}{NM}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}F(m,n)e^{j(2\pi/M)mi}e^{j(2\pi/N)nk}$$

其中，m=0，1，…，M-1；n=0，1，…，N-1。

从计算公式中可以看出，DFT的输入和输出均为离散值，非常适用于计算机的运算操

作，因此利用计算机进行傅立叶变换的通常形式为 DFT。

3. 可分离性

根据可分离性，以上二维傅立叶变换可由连续两次一维傅立叶变换，即

$$F(m,n)=\sum_{i=0}^{M-1}\left[\sum_{k=0}^{N-1}f(i,k)e^{-j(2\pi/N)nk}\right]e^{-j(2\pi/M)mi}$$

得到，从而将二维 DFT 分解为水平和垂直两部分运算，上式中方括号中的项表示在图像的行上计算的 DFT，方括号外边的求和则实现结果数组在列上 DFT。这种分解可使我们用一维的 FFT 来快速实现二维 DFT，二维逆 DFT 也可以同样分解(离)。

6.1.3 快速傅立叶变换(FFT)

通常计算一维 DFT 所需的乘法和加法操作的次数是 N^2 次，因为它把所有的复指数值都存在一张表中，这样的计算量实在太大；而快速傅立叶算法将 DFT 计算式分解，可以将操作降到(Nlb(N))数量级；尤其当 N 是 2 的幂(即 $N=2^p$，其中 p 是整数)时，计算效率最高，实现起来也最简单。

其算法思想是：先将原图像进行转置，按行对转置后的图像矩阵作一维 FFT，将此变换所得的中间矩阵再转置，再按行对转置后的中间矩阵作一维 FFT，最后得到的就是二维 FFT。

6.1.4 傅立叶变换的应用

傅立叶变换在 MATLAB 的图像处理中具有广泛应用，例如进行滤波去噪、边缘检测等。这些将在后续的章节中具体介绍，这里只举几个简单的例子。

1. 快速卷积

傅立叶变换的性质表中的卷积定理指出了傅立叶变换的一个主要好处，即与其在空域中作不直观的、难懂的卷积，不如在频域中作乘法，而且可以达到相同的效果。

根据卷积定理

$$F[A*B]=F[A]\bullet F[B]$$

则有

$$A*B=F^{-1}\{F[A]\bullet F[B]\}$$

下面的程序就是按上述原理实现快速卷积。首先构造两个矩阵 A 和 B，代码如下：

```
A = magic(3);    %用 1 到 9 之间的数产生一个 3×3 方阵
B = ones(3);     %生成一个全1的3×3方阵
A(8,8) = 0;      %用0将A补成8×8的方阵
B(8,8) = 0;      %用0将B补成8×8的方阵
```

然后，分别对 A 和 B 进行傅立叶变换，代码如下：

```
A2=fft2(A);
B2=fft2(B);
```

在频域内进行点乘，代码如下：

```
M= A2 * B2;
```

最后将点乘的结果变换回空域，并截取有效数据，代码如下：

```
C = ifft2(M);
C = C(1:5,1:5);
C = real(C);
```

最后的结果为

```
C =

    8.0000    9.0000   15.0000    7.0000    6.0000
   11.0000   17.0000   30.0000   19.0000   13.0000
   15.0000   30.0000   45.0000   30.0000   15.0000
    7.0000   21.0000   30.0000   23.0000    9.0000
    4.0000   13.0000   15.0000   11.0000    2.0000
```

2. 模板匹配

模板匹配也是模式识别的一种有效方法，具体方法是：以待定位的目标为模板在待识别图像上滑动，同时作连贯运算，对运算结果取适当的阈值，确定待定位的目标的位置。连贯操作就是首先将模板图像旋转 180°，然后进行快速卷积。下面的例子采用所介绍的方法对图像中的字母"a"进行定位(见图 6.16)，得到的就是相关运算的结果(见图 6.17)。

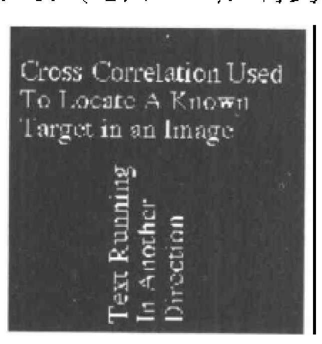
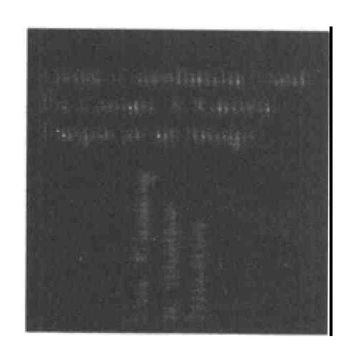


图 6.16 待识别图像和待匹配字符



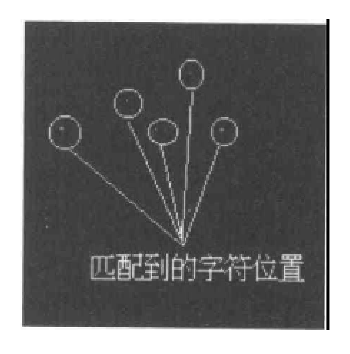


图 6.17 卷积的结果和文本匹配的结果

6.2 离散余弦变换

离散余弦变换，简称 DCT，是一种实数域变换，其变换核为实数的余弦函数，计算速度较快，而且对于具有一阶马尔柯夫过程的随机信号，DCT 十分接近于 Karhunen-Loeve 变换，也就是说它是一种近似最佳变换，很适于做图像压缩和随机信号处理。

对于数字图像 $X(m,n)$，$0\leqslant m\leqslant M, 0\leqslant n\leqslant N$，其二维 DCT 变换定义为

$$Y(k,l)=\alpha_k\alpha_l\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}X(m,n)\cos\frac{(2m+1)k\pi}{2M}\cos\frac{(2n+1)l\pi}{2N}$$

其中，$k=0,1,\cdots,M-1$；$l=0,1,\cdots,N-1$

$$\alpha_k=\begin{cases}1/\sqrt{M} & k=0\\ \sqrt{2/M} & 1\leqslant k\leqslant M-1\end{cases},\quad \alpha_l=\begin{cases}1/\sqrt{N} & l=0\\ \sqrt{2/N} & 1\leqslant l\leqslant N-1\end{cases}$$

二维 DCT 变换具有可分离性，可以分解为双重的一维 DCT，实现起来非常方便。

二维 DCT 反变换(IDCT)定义为

$$X(m,n)=\alpha_k\alpha_l\sum_{k=0}^{M-1}\sum_{l=0}^{N-1}Y(k,l)\cos\frac{(2m+1)k\pi}{2M}\cos\frac{(2n+1)l\pi}{2N}$$

其中，

$$\alpha_k=\begin{cases}1/\sqrt{M} & k=0\\ \sqrt{2/M} & 1\leqslant k\leqslant M-1\end{cases},\quad \alpha_l=\begin{cases}1/\sqrt{N} & l=0\\ \sqrt{2/N} & 1\leqslant l\leqslant N-1\end{cases}$$

由上式可知，原始图像 $X(m,n)$ 可表示为以 $Y(k,l)$ 为权值的一系列函数

$$\alpha_k\alpha_l\sum_{k=0}^{M-1}\sum_{l=0}^{N-1}\cos\frac{(2m+1)k\pi}{2M}\cos\frac{(2n+1)l\pi}{2N}$$

$$0\leqslant m\leqslant M-1,\quad 0\leqslant n\leqslant N-1$$

的加权组合，这组函数就是 DCT 基函数。

图 6.18 是用图像方式显示 8×8 DCT 基函数矩阵。

DCT 变换的实现有两种方法，一种是基于 FFT 的快速算法，这是通过工具箱提供的 dct2 函数实现的；另一种是 DCT 变换矩阵(transform matrix)方法。变换矩阵方法非常适合做 8×8 或 16×16 的图像块的 DCT 变换，工具箱提供了 dctmtx 函数来计算变换矩阵。一个 $M\times M$ 的 DCT 变换矩阵 T 定义为

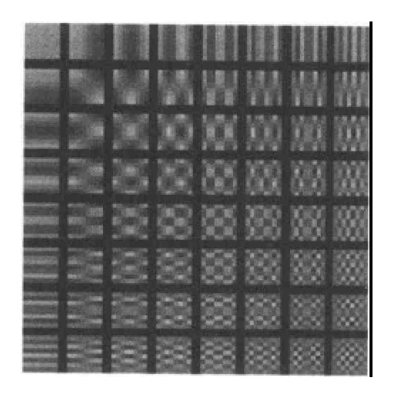


图 6.18 DCT 变换矩阵

$$T=(T_{i,j})=\begin{cases}1/\sqrt{M} & i=0,0\leqslant j\leqslant M-1\\ \sqrt{\frac{2}{M}}\cos\frac{\pi(2j+1)i}{2M} & 1\leqslant i\leqslant M-1,0\leqslant j\leqslant M-1\end{cases}$$

则 X 的 DCT 变换 Y 为 $Y=TXT^T$。

离散余弦变换在图像压缩中有很多应用，它是 JPEG、MPEG 等数据压缩标准的重要数学基础。在 JPEG 压缩算法中，首先将 RGB 分量转化成亮度分量和色差分量，同时丢失一半的色彩信息(空间分辨率减半)；然后将输入图像划分为8×8或16×16的图像块，对每个图像块做 DCT 变换；然后舍弃高频的系数，并对余下的系数进行量化以进一步减少数据量；最后使用 RLE 和 Huffman 编码来完成压缩任务。解压缩时首先对每个图像块做 DCT 反变换，然后将图像块拼接成一幅完整的图像。下面是用 DCT 变换做图像压缩的例子，原始图像和经压缩—解压后的图像分别如图 6.19 和图 6.20 所示。



图 6.19 原始图像



图 6.20 经压缩—解压后的图像

```
I = imread('cameraman.tif');    %读入图像
I = double(I)/255;
T = dctmtx(8);  %计算离散变换矩阵，返回结果为双精度型
B = blkproc(I,[8 8],'P1*x*P2',T,T');   %实现图像的显示块操作
mask =[1 1 1 1 0 0 0 0
       1 1 1 0 0 0 0 0
       1 1 0 0 0 0 0 0
       1 0 0 0 0 0 0 0
       0 0 0 0 0 0 0 0
       0 0 0 0 0 0 0 0
       0 0 0 0 0 0 0 0
       0 0 0 0 0 0 0 0];
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);
```

```
imshow(I), figure, imshow(I2)
```

6.3 Radon 变 换

图像的投影(Projection)就是图像在某一方向上的线积分，对于数字图像来说，也就是在该方向的累加求和。图 6.21 是一幅图像在 X、Y 轴上的投影。

所谓 Radon 变换，就是将原始图像变换为它在各个角度的投影表示。图像 f (x,y)在任一角度θ上投影定义为

$$R_\theta(x')=\int_{-\infty}^{\infty} f(x'\cos\theta-y'\sin\theta,x'\sin\theta-y'\cos\theta)dy'$$

其中

$$\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}\cos\theta & \sin\theta\\-\sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}$$

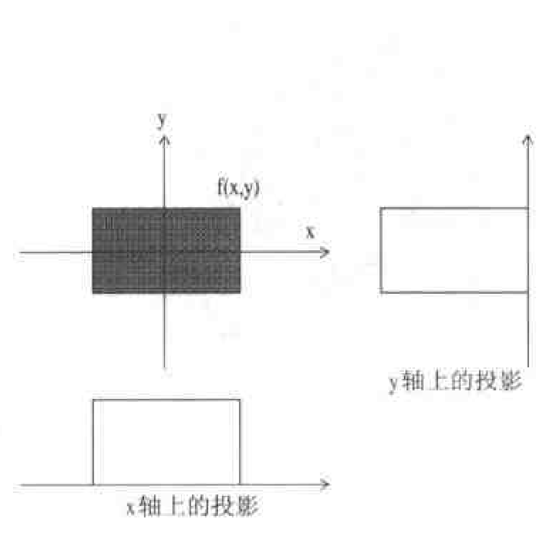


图 6.21 图像在坐标轴上的投影

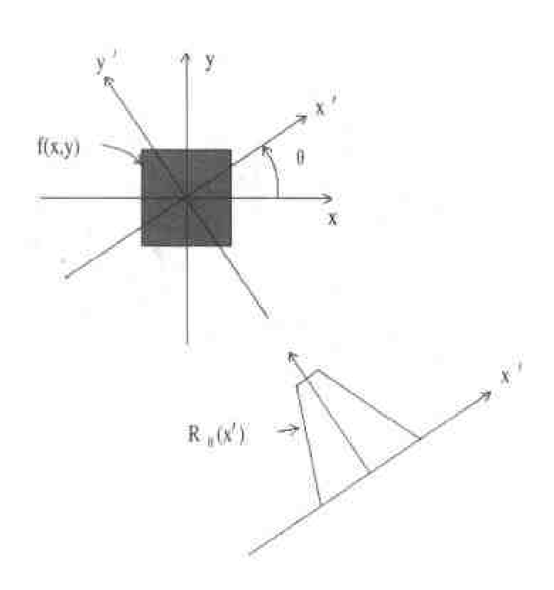


图 6.22 Radon 变换的几何关系

图 6.22 是 Radon 变换的几何关系图。

Radon 变换类似于计算机视觉中有名的 Hough 变换，可以用来检测图像中的直线，算法和程序如下。

原始图像如图 6.23 所示。

首先，提取图像的局部边缘(见图 6.24)。

```
I = imread('ic.tif');
BW = edge(I);
imshow(I)
figure, imshow(BW)
```

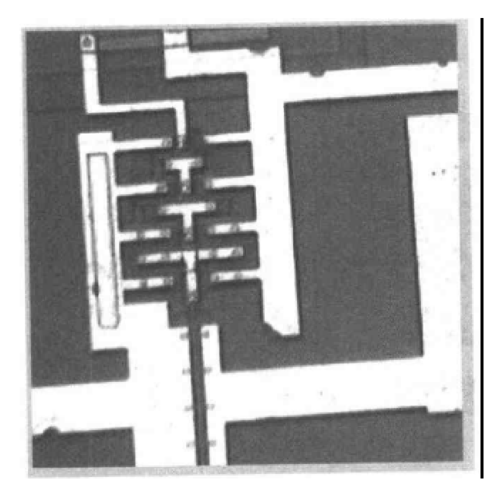




图 6.23　原始图像

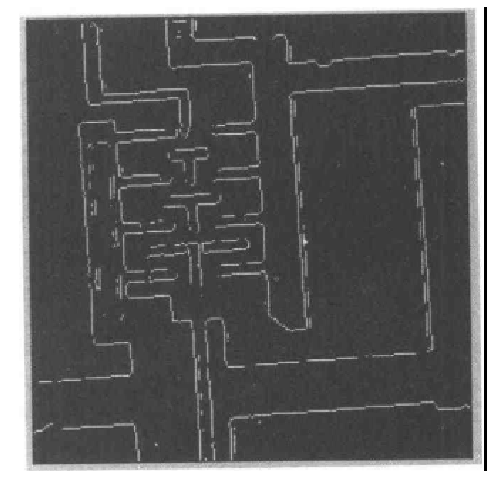


图 6.24　提取图像的局部边缘

然后，对边缘图像做 Radon 变换(见图 6.25)。

```
theta = 0:179;
[R,xp] = radon(BW,theta);
imagesc(theta, xp, R);
colormap(hot);
```

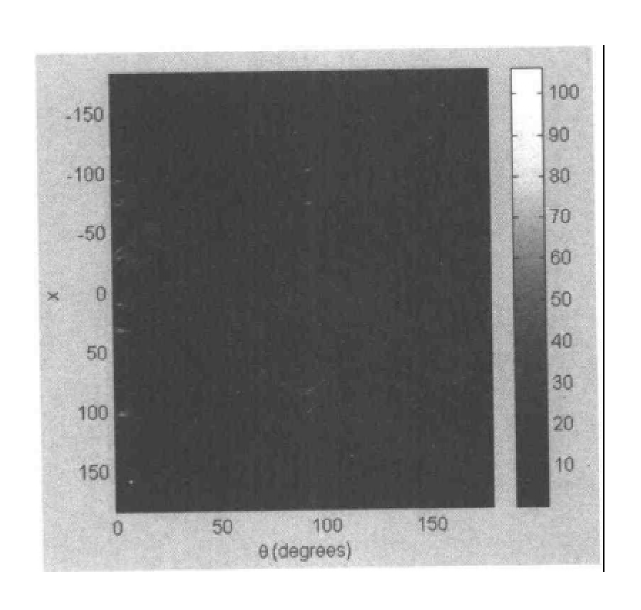


图 6.25　边缘图像的 Radon 变换

```
xlabel('\theta (degrees)');
ylabel('x');
colorbar;
```

最后，找出变换矩阵 R 的峰值，这些峰值对应着原图像上的直线。比如，Radon 变换矩阵在 x' = −101、θ = 94° 处有一个峰值，则对应原图在倾角 θ = 94° 的坐标轴 X' 的−101 处有一条垂直于 X'的直线(见图 6.26)。

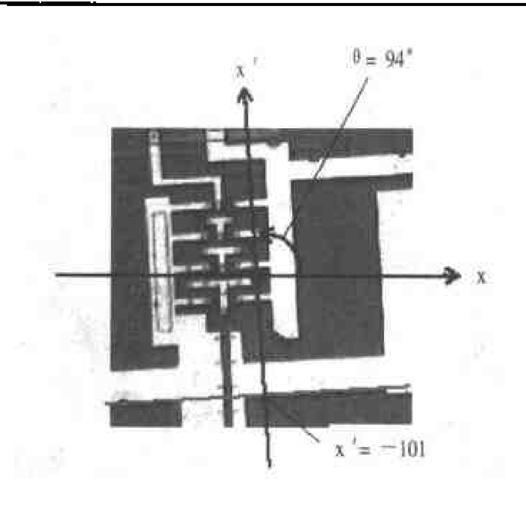


图 6.26　利用 Radon 变换提取的一条直线段

radon 函数的反变换通常应用于立体图像的剖面图分析。因为与 Radon 变换相反，所以 radon 函数的反变换可以用来利用投影数据重构原始图像。radon 函数是从平行光束投影中重构原始图像。



第七章　FIR 滤波器设计

FIR 滤波器就是有限脉冲响应滤波器，它具有以下优点：

(1) 易于表征为系数矩阵；

(2) 二维 FIR 滤波器是一维 FIR 滤波器的自然扩展；

(3) 有多种成熟的滤波器设计方法；

(4) 易于实现；

(5) 能够设计成具有严格的线性相位，而同时可以具有任意幅度特性，从而避免了图像处理中的变形；

(6) FIR 滤波器的单位抽样响应是有限长的，因而一定是稳定的。

对于图像处理来说，最常用的滤波器是二维 FIR 滤波器。因此本章中将介绍二维 FIR 滤波器各种设计方法：

(1) 窗口方法；

(2) 频率采样方法；

(3) 频率变换方法。

本章主要内容：

★ FIR 滤波器设计基础

★ 窗口方法

★ 频率采样法

★ 二维 FIR 滤波器设计的频率变换法

7.1　FIR 滤波器设计基础

FIR 滤波器的设计通常是在频域内进行的。为了分析 FIR 滤波器的频域特性，图像处理工具箱提供了计算二维线性系统频率响应的 freqz2 函数和计算期望频率响应矩阵的 freqspace 函数。

7.1.1　freqz2 函数

freqz2 函数语法如下：

```
[H,f1,f2] = freqz2(h,n1,n2)
[H,f1,f2] = freqz2(h,[n2 n1])
[H,f1,f2] = freqz2(h,f1,f2)
```

```
[H,f1,f2] = freqz2(h)
[...] = freqz2(h,...,[dx dy])
[...] = freqz2(h,...,dx)
freqz2(...)
```

其中：

[H,f1,f2] = freqz2(h,n1,n2)表示返回 FIR 滤波器 h 的 n1 × n2 点频率响应矩阵 H，n1、n2 的缺省值为 64。f1 和 f2 为对应的频率轴向量，取归一化频率[-1，1]，其中 1 对应 Nequist 频率（即采样率的一半）。h 以二维 FIR 滤波器的计算核的形式出现。

[H,f1,f2] = freqz2(h,[n2 n1]) 与[H,f1,f2] = freqz2(h,n1,n2)返回的结果一样。

[H,f1,f2] = freqz2(h,f1,f2)表示返回指定频率坐标（f1,f2）处的频率响应 H。

[H,f1,f2] = freqz2(h)是[n2 n1] = [64 64]时[H,f1,f2] = freqz2(h,[n2 n1])的特殊情况。

[...] = freqz2(h,...,[dx dy])表示用[dx dy]代替 h 中采样点之间的间隔，dx 确定 x-维上的间隔，dy 确定 y-维上的间隔，缺省的间隔为 0.5，对应着 2.0 的采样频率。

[...] = freqz2(h,...,dx)表示用 dx 来确定两个坐标上的采样间隔。

freqz2(...)表示无返回值，只生成二维幅频响应的网格图。

数据类型要求：输入矩阵可以是 double 型或任何整数类型的，所有其它的输入都必须是 double 型，所有的输出都是 double 型的。

例 7.1.1 已知二维滤波器 h 的计算核如下：

```
h=[0.1667 0.6667 0.1667;0.6667 -3.3333 0.6667;0.1667 0.6667 0.1667]
h =
    0.1667    0.6667    0.1667
    0.6667   -3.3333    0.6667
    0.1667    0.6667    0.1667
```

计算并显示 h 的 64×64 点的频率响应：

```
freqz2(h)
```

结果如图 7.1 所示。

7.1.2 freqspace 函数

freqspace 函数返回等间隔频率响应的隐含频率范围。因为滤波器设计函数 fsample2、fwind1 和 fwind2 都是基于期望频率响应矩阵来设计过滤器的，所以通常都要先调用 freqspace 函数。

freqspace 函数的语法如下：

```
[f1,f2] = freqspace(n)
[f1,f2] = freqspace([m n])
[x1,y1] = freqspace(...,'meshgrid')
f = freqspace(N)
f = freqspace(N,'whole')
```

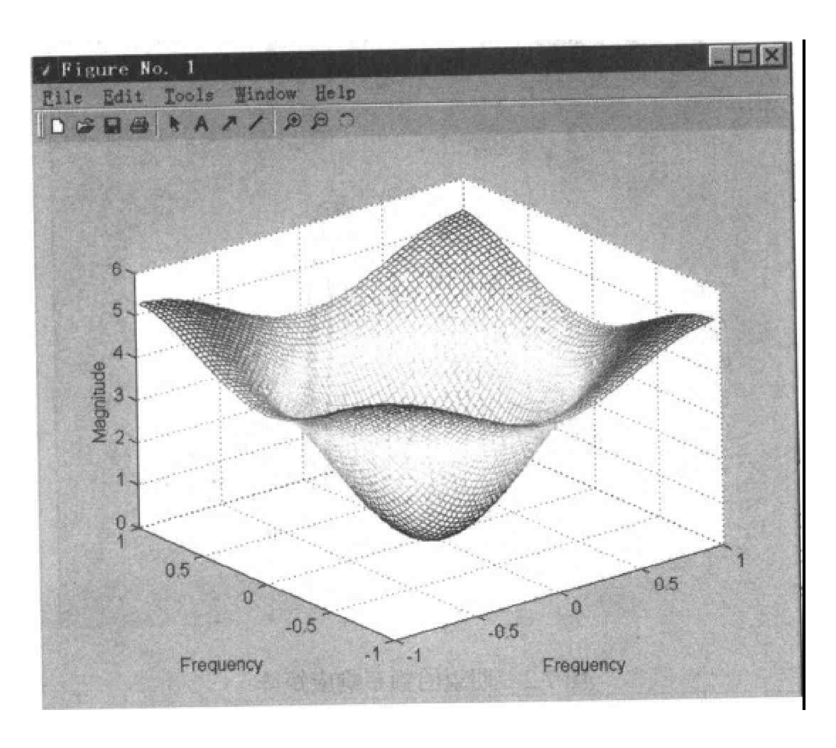


图 7.1　二维 FIR 滤波器的幅频响应

其中：

[f1,f2] = freqspace(n)表示对一个 n×n 矩阵，返回二维频率向量 f1 和 f2。当 n 为奇数时，f1 和 f2 都是 $[-n+1:2:n-1]/n$；当 n 为偶数时，f1 和 f2 都是 $[-n:2:n-2]/n$。

[f1,f2] = freqspace([m n])表示对一个 m×n 矩阵，返回二维频率向量 f1 和 f2。

[x1,y1] = freqspace(...,'meshgrid')等价于

[f1,f2] = freqspace(...);

[x1,y1] = meshgrid(f1,f2);

f = freqspace(N)表示假设 N 个均匀分布在单位圆上的点，返回一维频率向量 f。无论 N 是奇数或偶数，f 都是 $(0:2/N:1)$。当 N 是奇数时，返回 $(N+1)/2$ 个点；当 N 是偶数时，返回 $(N+2)/2$ 个点。

f = freqspace(N,'whole')表示返回 N 个均匀分布在整个单位圆上的点，在这种情况下，f 是 $0:2/N:2\times(N-1)/N$。

例 7.1.2　产生一个圆形的理想低通的频率响应。实现代码如下：

```
[f1,f2]=freqspace(25,'meshgrid');
Hd=zeros(25,25);
d=sqrt(f1.^2+f2.^2)<0.5;
Hd(d)=1;
mesh(f1,f2,Hd)
```

得到期望的频率响应矩阵如图 7.2 所示。

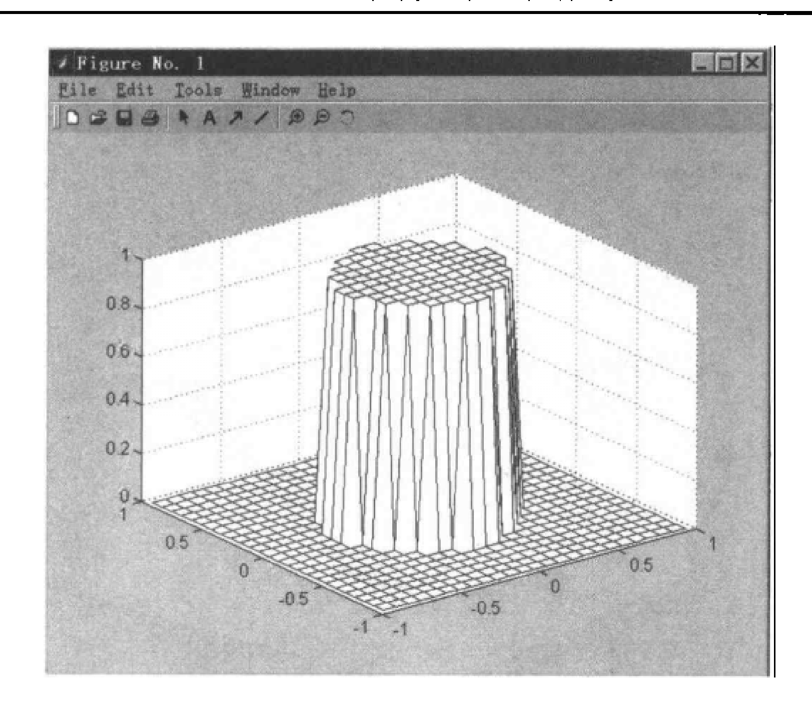


图 7.2 期望的频率响应矩阵

7.2 窗口方法

二维窗函数设计法是二维滤波器设计的最早技术之一，是一维窗函数方法的直接推广，它根据所要求的理想滤波器频率响应，设计一个 FIR 滤波器频率响应来逼近理想滤波器频率响应。

设要设计的二维 FIR 滤波器的单位脉冲响应为

$$h(n_1,n_2)=\omega(n_1,n_2)\cdot i(n_1,n_2)$$

其中，$i(n_1,n_2)$是无限冲激响应序列；$\omega(n_1,n_2)$是二维窗函数。等式两边做傅立叶变换，得

$$H(\omega_1,\omega_2)=\frac{1}{4\pi^2}\int\int I(\Omega_1,\Omega_2)W(\omega_1-\Omega_1,\omega_2-\Omega_2)\,d\Omega_1 d\Omega_2$$

上式表明，对无限冲激响应加窗的效果等价于在频域用 $W(\omega_1,\omega_2)$对 $I(\omega_1,\omega_2)$进行平滑。

常用的窗函数有矩形(Boxcar)窗、三角(Bartlett)窗、海明(Hamming)窗、汉宁(Hanning)窗、布莱克曼（Blackman）窗和凯泽（Kaiser）窗等。

窗口方法有两种：一种方法是用一维窗函数来构造二维窗函数，也就是使用函数 fwind1；可以将两个一维窗函数直接相乘，得到二维窗函数，或者将一维窗函数旋转得到二维窗函数。另一种方法是使用二维窗函数，也就是使用函数 fwind2。

7.2.1 fwind1 函数

fwind1 函数的语法：

h = fwind1(Hd,win)

h = fwind1(Hd,win1,win2)

h = fwind1(f1,f2,Hd,...)

其中：

h = fwind1(Hd,win)表示由期望的幅频响应 Hd，使用黄氏算法，从一维窗函数 win 得到近似圆对称的二维窗函数。矩阵 Hd 中的元素是在 x、y 频率轴的[-1.0,1.0]（在频率归一化中，1.0 对应着采样频率的一半或 π）区间内等间隔采样得到的点。为了得到精确的结果，可以使用 freqspace 函数返回的频率点来生成 Hd。窗函数 win 可以取以下值：boxcar、hamming、hanning、bartlett、blackman、kaiser 或 chebwin。如果 win 的长度是 n，那么 h 是 n×n 的。返回值 h 是计算核的形式，可以直接应用于 filter2 函数中。

h = fwind1(Hd,win1,win2)表示用两个一维窗函数 win1 和 win2 相乘生成可分离的二维窗。如果 win1 的长度是 n，win2 的长度是 m，那么 h 是 m×n 的。

h = fwind1(f1,f2,Hd,...) 指定了任意频率坐标(f1,f2)处的频率响应 Hd，f1 和 f2 都采用归一化频率。窗的长度控制着所得滤波器的尺寸。

数据类型要求：输入矩阵可以是 double 型或任何整数类型的；所有其它的输入必须是 double 型的；所有的输出都是 double 型的。

黄氏算法：

(1) 用一维窗函数生成近似圆对称的二维窗函数。

$$w(n_1,n_2)=w(t)\Big|_{t=\sqrt{n_1^2+n_2^2}}$$

其中，$w(t)$ 是一维窗函数，$w(n_1,n_2)$ 是得到的二维窗函数。

(2) 给定两个一维窗函数，生成可分离的二维窗函数。

$$w(n_1,n_2)=w_1(n_1)w_2(n_2)$$

fwind1 以 Hd 和二维窗函数为参数调用 fwind2，fwind2 先对 Hd 进行逆傅立叶变换，并与二维窗函数相乘，得到 h。

$$h_d(n_1,n_2)=\frac{1}{(2\pi)^2}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi}H_d(w_1,w_2)e^{jw_1n_1}e^{jw_2n_2}dw_1dw_2$$

$$h(n_1,n_2)=h_d(n_1,n_2)w(n_1,n_2)$$

例 7.2.1 分别采用 Hamming 窗、Bartlett 窗、Hanning 窗和 Blackman 窗设计近似圆对称的带通滤波器，通频带为[0.1,0.5]。

首先，创建包含了期望的带通响应的矩阵 Hd。代码如下：

```
[f1,f2] = freqspace(21,'meshgrid');
Hd = ones(21);
r = sqrt(f1.^2 + f2.^2);
Hd((r<0.1)|(r>0.5)) = 0;
colormap(jet(64))
mesh(f1,f2,Hd)
```

期望的频率响应矩阵 Hd 如图 7.3 所示。

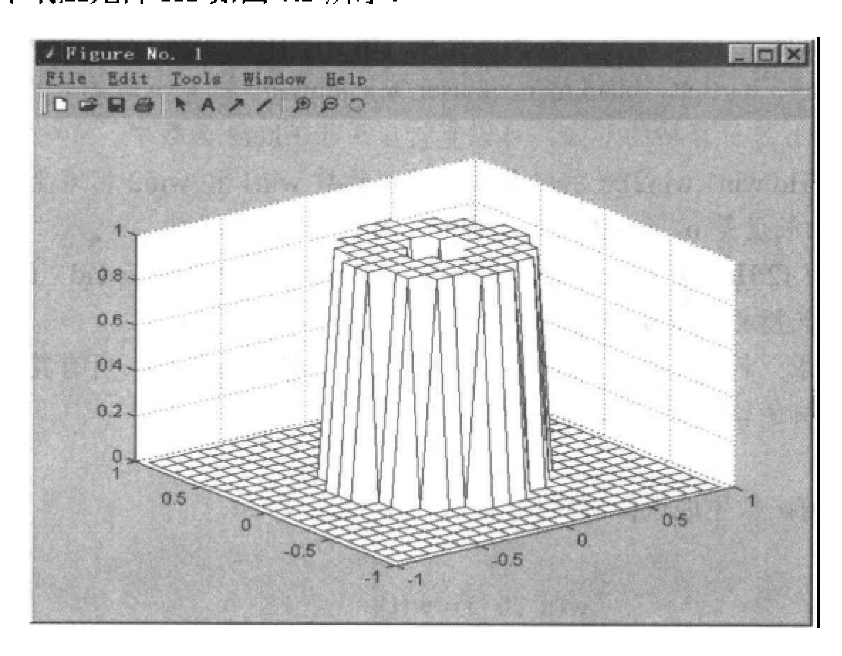


图 7.3　期望的频率响应矩阵 Hd

然后，分别用一维 Hamming 窗、Bartlett 窗、Hanning 窗和 Blackman 窗设计滤波器。代码如下：

```
h1 = fwind1(Hd,hamming(21));
figure,freqz2(h1)
h2 = fwind1(Hd, bartlett (21));
figure,freqz2(h2)
h3 = fwind1(Hd, hanning (21));
figure,freqz2(h3)
h4 = fwind1(Hd, blackman (21));
figure,freqz2(h4)
```

四个滤波器的频率响应分别如图 7.4～图 7.7 所示。

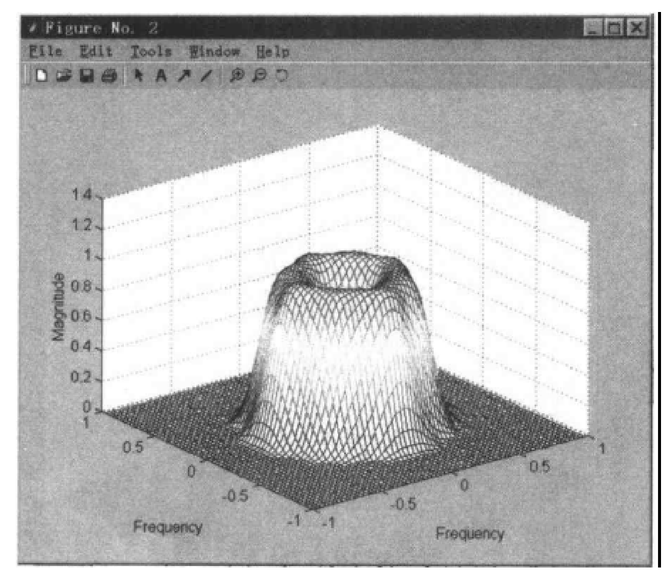


图 7.4　用 Hamming 窗设计的带通滤波器幅频响应

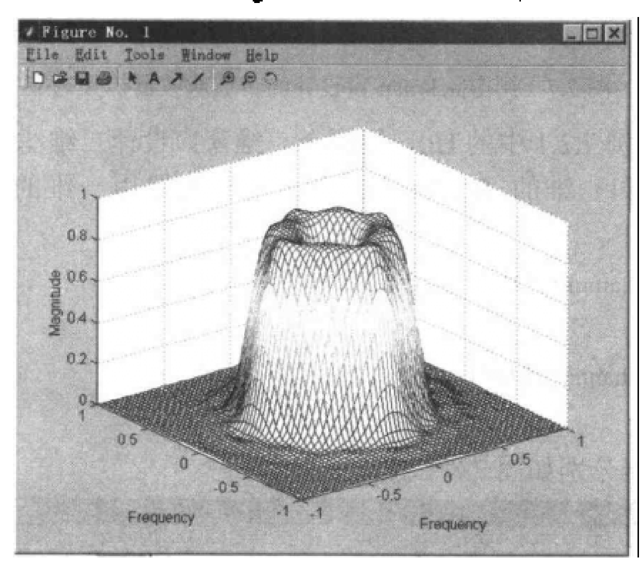


图 7.5　用 Bartlett 窗设计的带通滤波器幅频响应

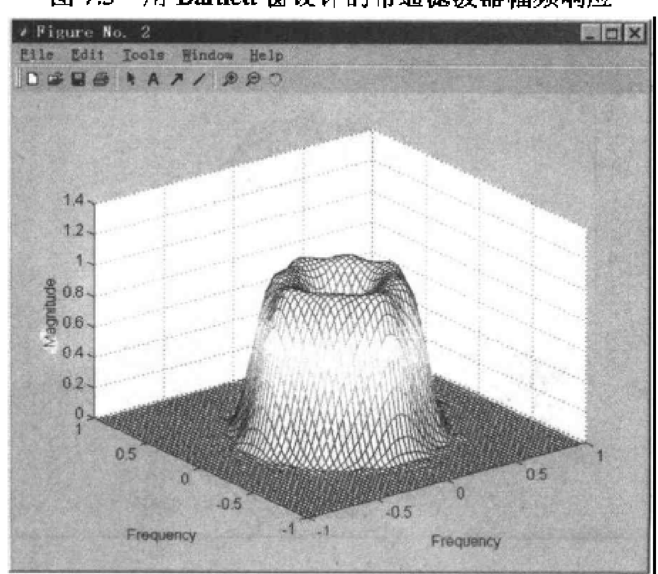


图 7.6　用 Hanning 窗设计的带通滤波器幅频响应

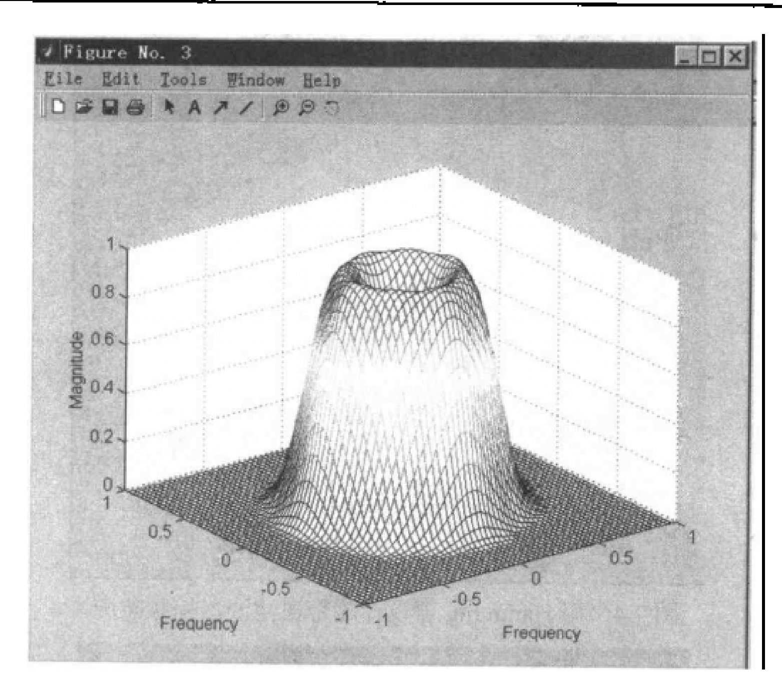


图 7.7　用 Blackman 窗设计的带通滤波器幅频响应

例 7.2.2　仍使用例 7.2.1 中的 Hd，用两个一维窗口设计二维滤波器。第一个滤波器用一维的 Hamming 窗和一维的 Bartlett 窗,第二个滤波器用一维的 Hanning 窗和一维的 Blackman 窗。代码如下：

```
h5= fwind1(Hd,hamming(21), bartlett(21));
figure,freqz2(h5)
h6 = fwind1(Hd,hanning(21),blackman(21));
figure,freqz2(h6)
```

得到的带通滤波器分别如图 7.8 和图 7.9 所示。

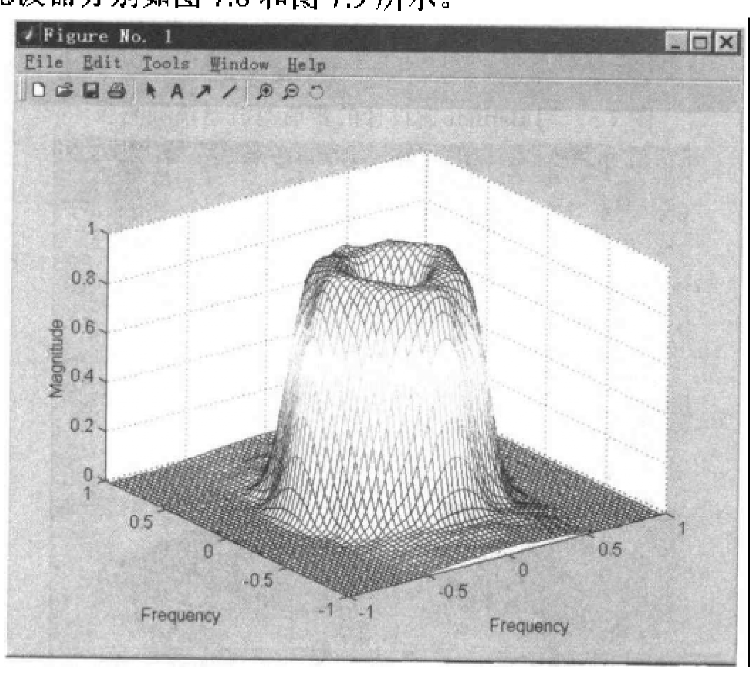


图 7.8　用 Hamming 窗和 Bartlett 窗设计的带通滤波器幅频响应

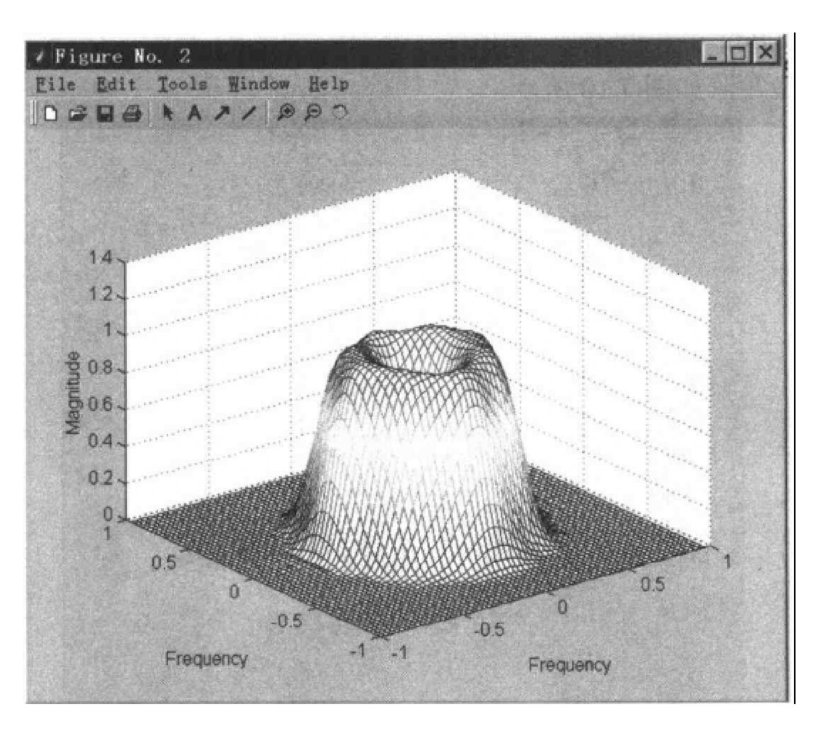


图 7.9　用 Hanning 窗和 Blackman 窗设计的带通滤波器幅频响应

7.2.2　fwind2 函数

fwind2 函数的语法：

h = fwind2(Hd,win)

h = fwind2(f1,f2,Hd,win)

h = fwind2(Hd,win)生成二维 FIR 滤波器的方法是：先对期望频率响应 Hd 进行逆傅立叶变换，然后与窗函数 win 相乘。矩阵 Hd 包含在迪卡尔平面上等间隔点处的期望频率响应；返回值 h 是计算核的形式，可以直接用于 filter2 函数中，h 的尺寸等于 win 的尺寸。为了得到精确的结果，可以使用 freqspace 函数返回的频率点来生成 Hd。

h = fwind2(f1,f2,Hd,win) 指定了 x、y 轴上任意频率坐标(f1,f2)处的幅频响应 Hd。频率向量 f1 和 f2 应该在[-1.0,1.0]的范围内，其中 1.0 对应着采样频率的一半，或 π 弧度。h 的尺寸等于 win 的尺寸。

数据类型要求：输入矩阵可以是 double 型的，或任何整数类型的；所有其它的输入都必须是 double 型的；所有的输出都是 double 型的。

例 7.2.3　用二维高斯窗设计近似圆对称的带通滤波器，通频带为[0.1,0.5]。

首先，创建包含了期望的带通响应的矩阵 Hd。代码如下：

```
[f1,f2] = freqspace(21,'meshgrid');
Hd = ones(21);
r = sqrt(f1.^2 + f2.^2);
Hd((r<0.1)|(r>0.5)) = 0;
colormap(jet(64));
```

```
mesh(f1,f2,Hd);
```

期望频率响应矩阵如图 7.10 所示。

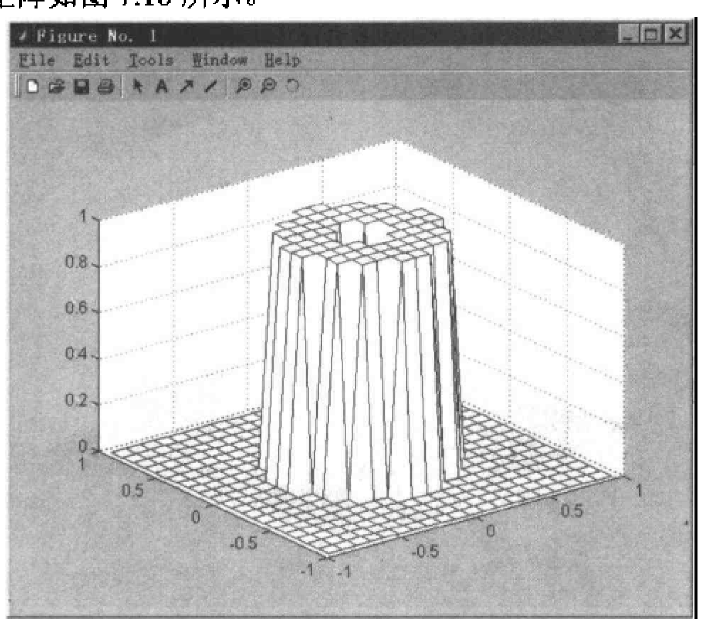


图 7.10 期望频率响应矩阵的幅频响应

然后，生成旋转对称的二维高斯窗函数。代码如下：

```
win = fspecial('gaussian',21,2);
win = win ./ max(win(:));  %将窗口的最大值归一化
mesh(win)
```

二维高斯窗函数如图 7.11 所示。

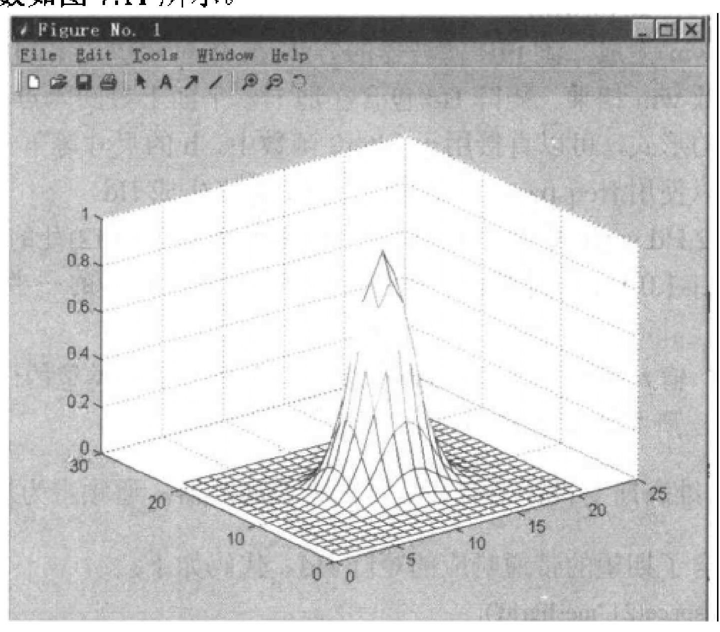


图 7.11 二维高斯窗函数

最后，用前一步生成的窗函数设计滤波器。代码如下：

```
h1 = fwind2(Hd,win);
freqz2(h1)
```

用二维窗函数法设计的带通滤波器的频率响应如图 7.12 所示。

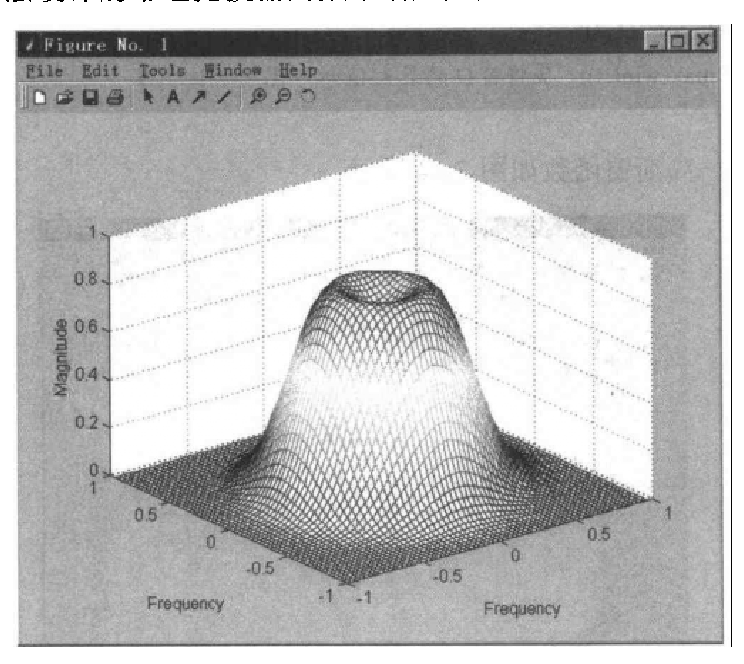


图 7.12 用二维窗函数法设计的低通滤波器的频率响应

例 7.2.4 用二维拉普拉斯—高斯窗设计近似圆对称的高通滤波器。

首先，创建包含了期望的带通响应的矩阵Hd。代码如下：

```
[f1,f2] = freqspace(21,'meshgrid');
Hd = ones(21);
r = sqrt(f1.^2 + f2.^2);
Hd(r<0.5) = 0;
colormap(jet(64))
mesh(f1,f2,Hd)
```

期望频率响应矩阵如图7.13所示。

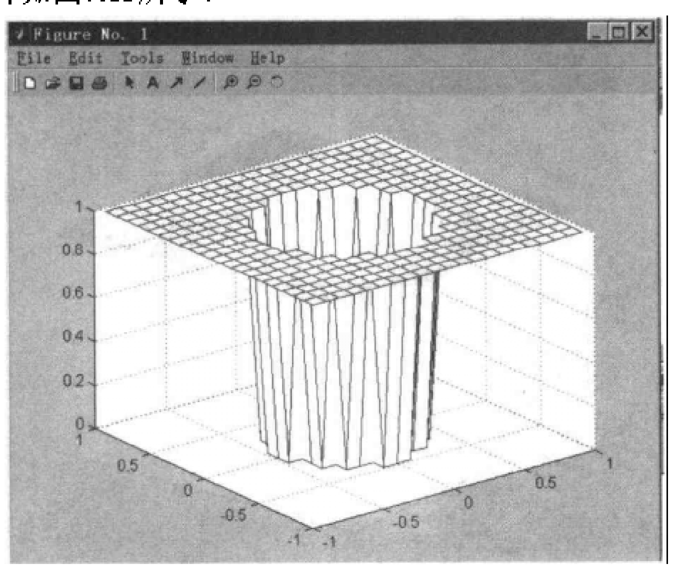


图 7.13 期望频率响应矩阵的频率响应

然后，生成旋转对称的二维拉普拉斯—高斯窗函数。代码如下：

```
win = fspecial('log',21,2);
win = win ./ max(win(:));  %将窗口的最大值归一化
mesh(win)
```

二维拉普拉斯—高斯窗函数如图 7.14 所示。

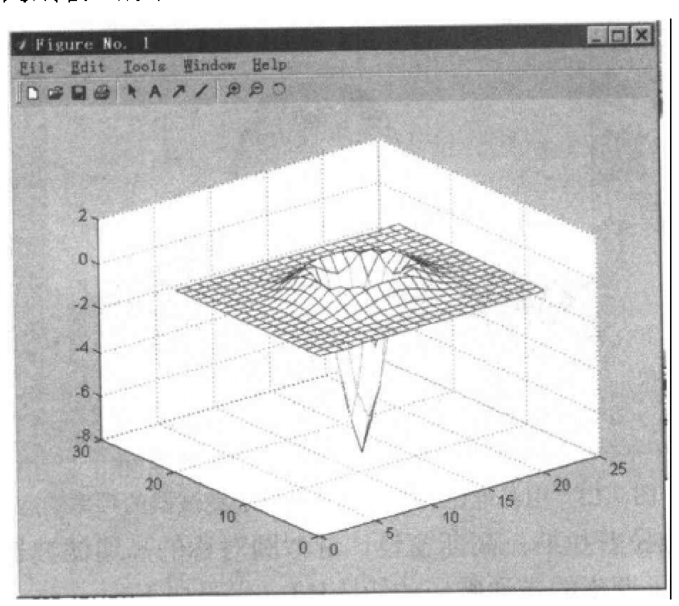


图 7.14　二维拉普拉斯—高斯窗函数

最后，用前一步生成的窗函数设计滤波器，代码如下：

```
h2= fwind2(Hd,win);
freqz2(h2)
```

用二维窗函数法设计的高通滤波器的频率响应如图 7.15 所示。

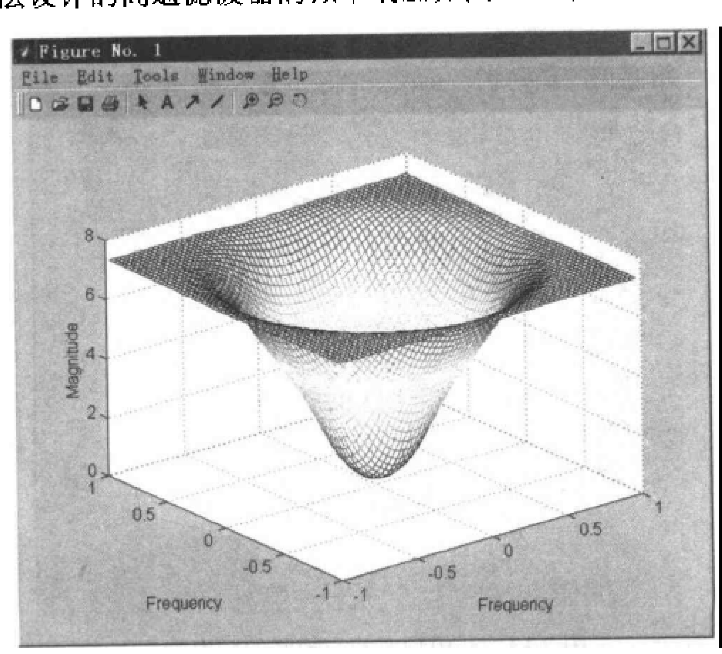


图 7.15　用二维窗函数法设计的高通滤波器的频率响应

7.3 频率采样法

在频率采样设计法中，滤波器的频率响应指标是由连续频率响应的采样值得到的，用其中非零频率响应的采样值就可计算冲激响应矩阵。

设期望的频率响应矩阵为 h(m,n), 0≤m≤M；0≤n≤N，其频率响应为

$$H(e^{j\omega_1},e^{j\omega_2})=\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}h(m,n)e^{-j(\omega_1 m+\omega_2 n)}$$

频域的采样为

$$H(k,l)=\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}h(m,n)e^{-j2\pi(\frac{mk}{M}+\frac{nl}{N})}$$

则所设计的冲激响应矩阵为

$$h(m,n)=\frac{1}{MN}\sum_{k=0}^{M-1}\sum_{l=0}^{N-1}H(k,l)e^{j2\pi(\frac{mk}{M}+\frac{nl}{N})}$$

对于线性相位的二维频率采样滤波器，还可通过 DFT 系数的对称关系进一步简化计算过程。

在图像处理工具箱中，用频率采样法设计二维滤波器的函数是 fsamp2。

fsamp2 函数的语法：

h = fsamp2(Hd)

h = fsamp2(f1,f2,Hd,[m n])

其中：

h = fsamp2(Hd) 由期望的频率响应矩阵 Hd 用频域采样法设计二维 FIR 滤波器，滤波器的系数返回到 h 中，h 是计算核的形式，可以直接应用于 filter2 函数中。h 的大小与 Hd 相同。矩阵 Hd 包含期望的频率响应采样点，这些采样点是在 x、y 频率轴的[-1.0,1.0]区间上等间隔采样得到的。为了得到精确的结果，可以使用 freqspace 函数返回的频率点生成 Hd。

h = fsamp2(f1,f2,Hd,[m n]) 指定了频率坐标(f1,f2)处的频率响应 Hd，f1、f2 采用相对频率。得到的滤波器在最小平方意义上尽可能地逼近期望响应。为了得到最好的结果，至少要有 m×n 个期望频率点，如果所指定的点数少于 m×n 个，则会提出警告。[m n]指定了 h 矩阵大小为 m×n。

数据类型要求：输入矩阵可以是 double 型的，或任何整数类型的；所有其它的输入都必须是 double 型的；所有的输出都是 double 型的。

例 7.3.1　用频率采样法设计二维低通滤波器。

首先，创建包含了期望的频率响应的矩阵Hd。代码如下：

```
[f1,f2] = freqspace(21,'meshgrid');
Hd = zeros(21);
Hd(7:15,7:15) = 1;
Axis([-1 1 -1 1 0 1.2]);
```

```
colormap(jet(64))
mesh(f1,f2,Hd)
```

期望的滤波器的频率响应如图 7.16 所示。

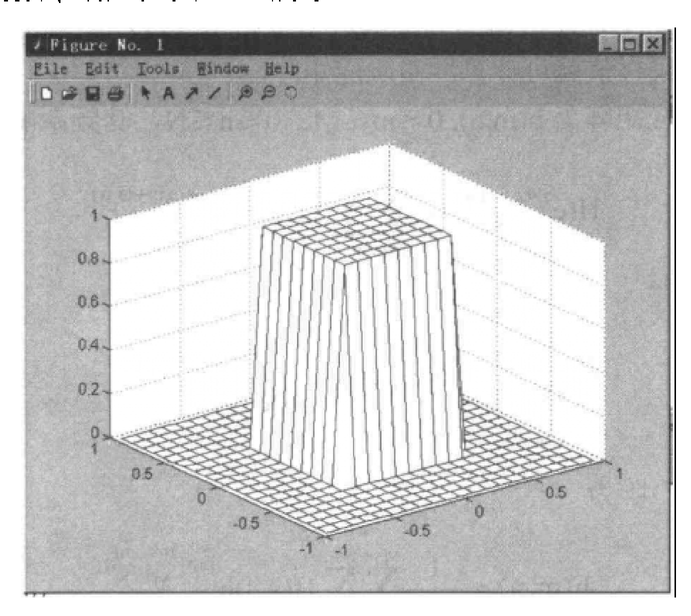


图 7.16 期望的滤波器频率响应图

然后，设计相应的滤波器。代码如下：

```
h = fsamp2(Hd);
figure,freqz2(h)
```

用频域采样法设计的滤波器的频率响应如图7.17所示。比较期望频率响应与实际的滤波器的频率响应，可以看到明显的波纹。这些波纹是频率采样设计方法中的重要问题，出现在期望的响应中较明显的过渡中。读者可以通过使用较大的滤波器来降低波纹的空间区域，但并不能降低波纹的高度，而且需要更多的滤波计算时间。为了得到期望频率响应的更加光滑的近似值，可以考虑使用频率变换方法或窗口方法。

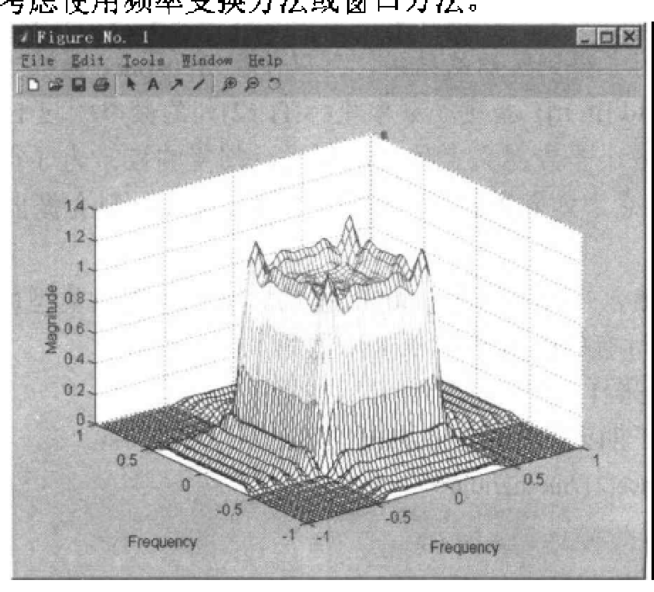


图 7.17 频域采样法设计的滤波器的频率响应

7.4 二维 FIR 滤波器设计的频率变换法

频率变换方法通过频率变换，将一个一维零相位 FIR 滤波器变换成二维滤波器。由于它保持了一维滤波器的大多数特性（如传输带宽、纹波特性等），因而如果一维滤波器是最优的，则变换得到的二维滤波器也容易保持最优性。另外，通过频率变换得到的二维滤波器具有有效的实现结构，比直接用卷积法或用 DFT 实现的运算量要小。

MATLAB 图像处理工具箱提供了函数 ftrans2，用于实现频率变换算法。

ftrans2函数的语法：

```
h = ftrans2(b,t)
h = ftrans2(b)
```

其中：

h = ftrans2(b,t)由一个一维 FIR 滤波器的冲激响应 b 通过频率变换 t 生成二维滤波器 h。t 为变换矩阵，缺省地采用 McClellan 变换矩阵，即

$$t=\frac{1}{8}\begin{bmatrix}1 & 2 & 1\\ 2 & -4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

b 必须是一维的长度为奇数（第一类）的 FIR 滤波器，例如，可以由 fir1 函数、fir2 函数或 remez 函数得到。返回值 h 是计算核的形式，可以直接用于 filter2 函数；如果 t 是 m×n 维的，b 的长度是 Q，那么 h 的尺寸是

$$\left(\frac{(m-1)\times(Q-1)}{2}+1\right)\times\left(\frac{(n-1)\times(Q-1)}{2}+1\right)$$

h = ftrans2(b) 使用缺省的 McClellan 变换矩阵。

ftrans2 返回的二维滤波器的频率响应由以下变换定义：

$$H(w_1,w_2)=B(w)\Big|_{\cos w=T(w_1,w_2)}$$

其中 B(w) 是一维滤波器 b 的傅立叶变换，即

$$B(w)=\sum_{n=-N}^{N}b(n)e^{-jwn}$$

而 $T(w_1,w_2)$ 是变换矩阵 t 的傅立叶变换，即

$$T(w,w)=\sum_{n_2}\sum_{n_1}t(n_1,n_2)e^{-jw_1n_1}e^{-jw_2n_2}$$

最后返回的滤波器 h 是 $H(w_1,w_2)$ 的逆傅立叶变换：

$$h(n_1,n_2)=\frac{1}{(2\pi)^2}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi}H(w_1,w_2)e^{jw_1n_1}e^{jw_2n_2}dw_1dw_2$$

例 7.4.1 用频率变换法设计近似圆对称的二维 FIR 低通滤波器。

首先，创建一个一维 FIR 低通滤波器，变换矩阵是可以根据需要定义的，这里定义了一个截断频率为 0.6 的十阶低通滤波器和一个截断频率为 0.5 的十四阶低通滤波器，代码如下：

```
b1=remez(10,[0 0.4 0.6 1],[1 1 0 0]);
[H1,w]=freqz(b1,1,64,'whole');
colormap(jet(64))
plot(w/pi-1,fftshift(abs(H1)))
b2=remez(14,[0 0.3 0.5 1],[1 1 0 0]);
[H2,w]=freqz(b2,1,64,'whole');
colormap(jet(64))
plot(w/pi-1,fftshift(abs(H2)))
```

用于频率变换的两个一维滤波器的频率响应如图 7.18 和图 7.19 所示。

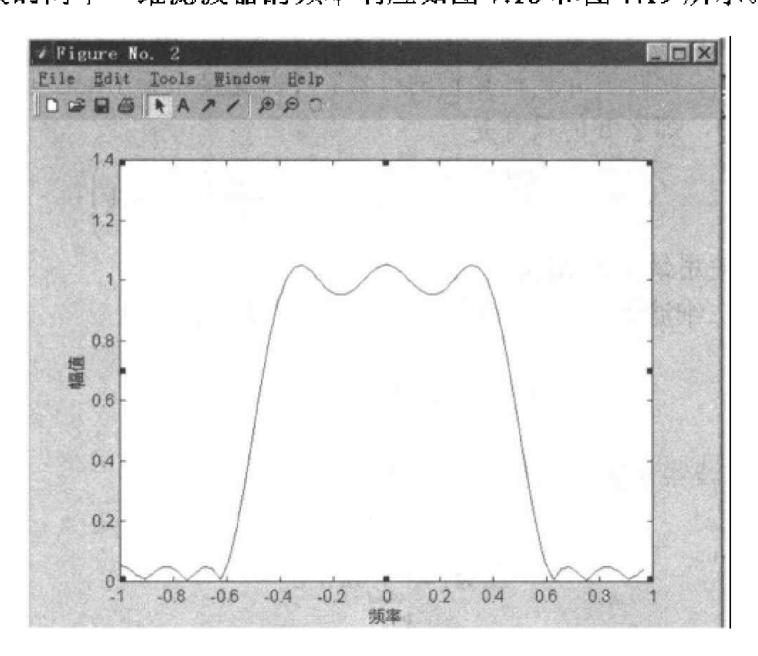


图 7.18 截断频率为 0.6 的十阶低通滤波器频率响应

然后，用缺省的 McClellan 变换生成期望的近似圆对称的滤波器。代码如下：

```
h1=ftrans2(b1);
figure,freqz2(h1,[32 32])
h2=ftrans2(b2);
figure,freqz2(h2,[32 32])
```

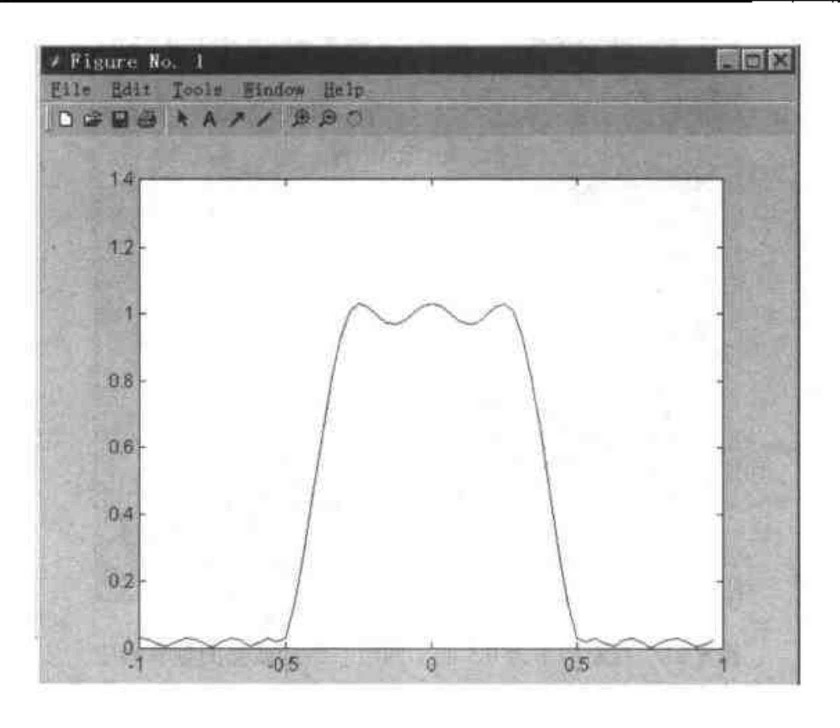


图 7.19　截断频率为 0.5 的十四阶低通滤波器的频率响应

两个二维低通滤波器的频率响应如图 7.20 和图 7.21 所示。

本例中为了显示差别，只任意设计了两个截断频率和阶数都不同的低通滤波器。在实际应用中，阶数和截断频率都要根据实际情况而定，读者可以通过多次试验得到比较理想的滤波器。

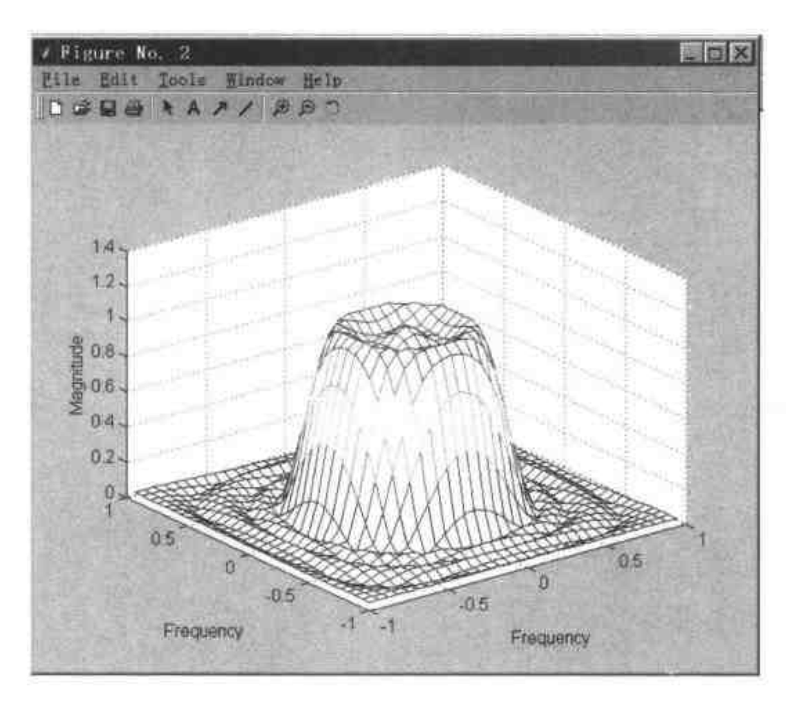


图 7.20　截断频率为 0.6 的十阶二维滤波器的频率响应

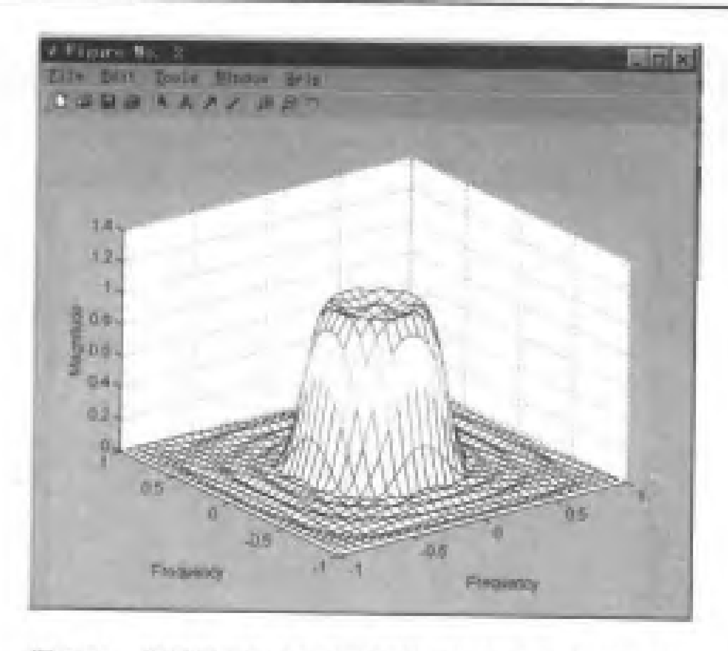


图 7.21　截断频率为 0.5 的十四阶二维滤波器的频率响应

图像处理技术篇

第八章 图 像 增 强

图像增强是一类基本的图像处理技术，其目的是对图像进行加工，以得到对具体应用来说视觉效果更“好”、更“有用”的图像。这里的“好”和“有用”因具体的应用目的和要求而异，并且所需要的具体增强技术也可以大不相同。

目前常用的增强技术根据对图像进行处理所在的空间不同，可分为基于图像域的方法和基于变换域的方法。第一类，直接在图像所在的空间进行处理，也就是在像素组成的空间里直接对像素进行操作；第二类，在图像的变换域对图像进行间接处理。另外，在第一类方法里还可分为两组：一是基于像素(点)的，也就是对图像的每次处理是对每个像素进行的，增强过程对每个像素的处理与其它像素无关；二是基于模板的，也就是对图像的每次处理是对小的子图像(模板)进行的。

空域增强方法可表示为

$$g(x,y)=EH[f(x,y)] \tag{8.1}$$

其中，$f(x,y)$和$g(x,y)$分别为增强前后的图像，EH代表增强操作。如果EH是定义在每个(x,y)上的，则EH是点操作；如果EH是定义在(x,y)的某个邻域上的，则(x,y)成为模板操作。如果模板最小，则可以是一个像素点，此时就变成了点操作，所以点操作可看作是模板操作的一个特例。

本章将分别介绍利用MATLAB实现以上所提到的各种图像处理技术。

本章主要内容：

★ 空域变换增强
★ 空域滤波增强
★ 频域增强

8.1 空域变换增强

当$g(x,y)$的值取决于在(x,y)处的$f(x,y)$值时，EH就是一个灰度变换。如以s和t分别表示$f(x,y)$和$g(x,y)$在(x,y)位置处的灰度值，则此时式(8.1)可写成

$$t=EH(s) \tag{8.2}$$

基于点操作的方法也叫灰度变换，常见的方法有以下几类：

(1) 直接对每个像素进行操作；

(2) 借助直方图进行变换；

(3) 借助对一系列图像间的操作进行变换。

8.1.1 直接灰度调整

1. 增强对比度

增强对比度实际是增强原图的各部分的反差。实际中往往是通过增加原图中某两个灰度值之间的动态范围来实现的。原始图像如图 8.1 所示，典型的增强对比度的变换曲线如图 8.2 所示。



图 8.1 原始图像

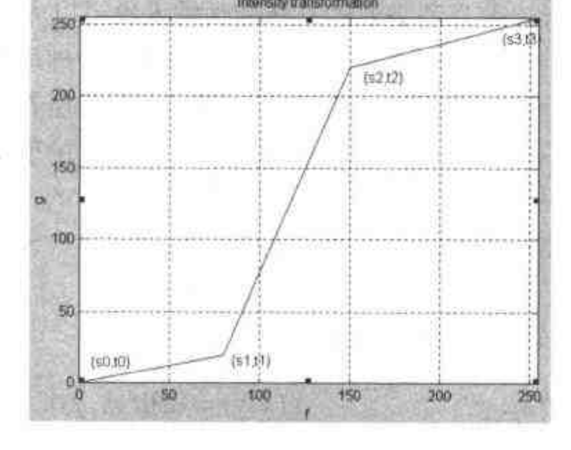


图 8.2 增强对比度的变换曲线

从图中可以看出，通过这样一个变换，原图中灰度值在 0～s1 和 s2～255 的动态范围减小了，而原图中灰度值在 s1～s2 的动态范围增加了，从而使这个范围内的对比度增加了。在实际应用中，s1、s2、t1、t2 可取不同的值进行组合，从而可得到不同的效果。

例 8.1.1 应用变换曲线增强对比度。代码如下：

```
X1=imread('pout.tif ');
figure,imshow(X1)
f0=0;g0=0;
f1=70;g1=30;
f2=180;g2=230;
f3=255;g3=255;
%绘制变换函数曲线
figure,plot([f0,f1,f2,f3],[g0,g1,g2,g3])
axis tight,xlabel('f'),ylabel('g')
title('intensity transformation')

r1=(g1-g0)/(f1-f0);
```

```
    b1=g0-r1*f0;
    r2=(g2-g1)/(f2-f1);
    b2=g1-r2*f1;
    r3=(g3-g2)/(f3-f2);
    b3=g2-r3*f2;
    [m,n]=size(X1);
    X2=double(X1);
    for i=1:m    %循环对矩阵中的每个元素进行变换处理
       for j=1:n
            f=X2(i,j);
            g(i,j)=0;
            if (f>=0)&(f<=f1)
            g(i,j)=r1*f+b1;
            elseif (f>=f1)&(f<=f2)
            g(i,j)=r2*f+b2;
            elseif (f>=f2)&(f<=f3)
            g(i,j)=r3*f+b3;
            end
      end
    end
figure,imshow(mat2gray(g))
```

图 8.3 和图 8.4 分别是在所选的参数模式下所得的图像和变换曲线。在处理过的图片上可以看出，小孩衣服上的图标和衣服之间的对比度增强了，更容易辨认了。通常在进行特定任务的模式识别工作之前，要对所要识别的图像进行增强对比度的处理，以使辨识工作更容易，辨识结果更准确。对于不同的图像，根据不同的情况，所选的参数可能有所不同，要根据实际情况而定。



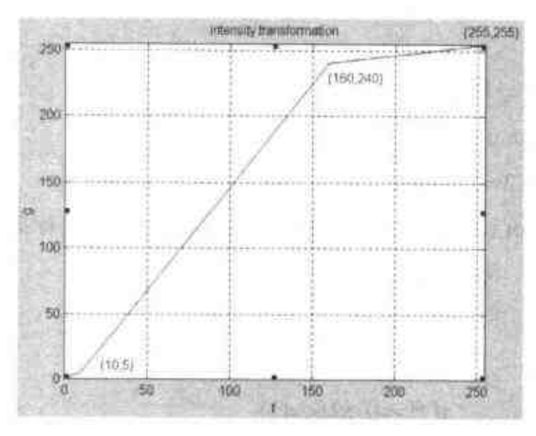



图 8.3　增强对比度所得的图像

图 8.4　增强对比度的变换曲线

例 8.1.2　利用 MATLAB 工具箱中的函数 imadjust()增强对比度。代码如下：

```
X1=imread('pout.tif ');
figure,imshow(X1)
J = imadjust(X1,[0.25, 0.6],[],1.2322)
figure, imshow(J)
```

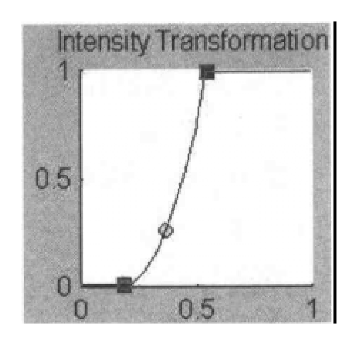


图 8.5　调用 MATLAB 工具箱中的函数增强对比度的变换曲线

采用以上参数的变换曲线如图 8.5 所示。

说明：J = imadjust(I,[low high],[bottom top],gamma) 表示返回图像 I 经调整后的图像 J。[low high]为原图像中要变换的灰度范围，[bottom top]指定了变换后的灰度范围，其中 low、high、bottom、top 的取值范围都是 0～1，并且 low<high，bottom<top，这是因为在 MATLAB 中将 0～255 的灰度范围映射到 0～1 的范围内。gamma 为矫正量 γ,它指定了变换曲线的形状,描述了 I 和 J 的值之间的关系。通常当 gamma 大于 1 时，图像变暗，而当 gamma 小于 1 时，图像变亮。上例中 gamma 所取的值只是个试验值，在这个取值下所得到的图像效果相对较好，读者根据需要可以取更大的值进行试验。

2. 图像求反

对图像求反是将原图的灰度值翻转，简单地说，就是使黑变白，使白变黑，普通的黑白底片和照片就是这样的关系。具体的变换就是将图像中每个像素的灰度值根据变换曲线进行映射。

例 8.1.3　应用变换曲线对图像求反。代码如下：

```
X1=imread('pout.tif ');
figure,imshow(X1)
f1=200;  %f1和g1分别表示变换曲线在纵轴和横轴上的最大范围
g1=256;
figure,plot([0,f1],[g1,0])
axis tight,xlabel('f'),ylabel('g')
title('intensity transformation')


k=g1/f1;
[m,n]=size(X1);
X2=double(X1);
for i=1:m    %循环对矩阵中的每个元素进行变换处理
   for j=1:n
        f=X2(i,j);
        g(i,j)=0;
        if (f>=0)&(f<=f1)
             g(i,j)=g1-k*f;
        else
             g(i,j)=0;
```

```
        end
    end
end
figure,imshow(mat2gray(g))
```

运行所得结果如图 8.6 所示，图像求反的变换曲线如图 8.7 所示。



图 8.6 图像求反所得的图像

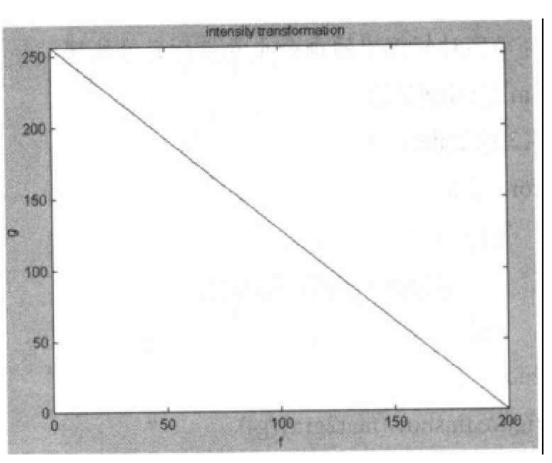


图 8.7 图像求反的变换曲线

3. 动态范围压缩

这种方法的目标与增强对比度相反。当原图的动态范围太大，超出了某些显示设备所允许的动态范围时，如果直接使用原图，则有一部分信息可能丢失。解决的办法是对原图进行灰度压缩。

例 8.1.4 采用对数形式(在 MATLAB 中用 log 表示自然对数)的变换函数进行动态范围压缩。

$$g = c * \log(1+f)$$

其中，c 是比例尺常数，见图 8.8 中所示的变换曲线。图 8.9 为原始图像。

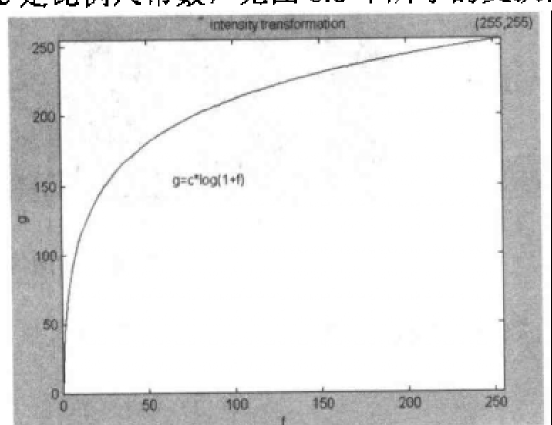


图 8.8 对数变换曲线

图 8.9 原始图像

运行如下代码段实现对数变换:

```
X1=imread('circuit.tif ');
figure,imshow(X1)
%绘制变换函数曲线
```

```
c=255/log(256);
x=0:1:255;
y=c*log(1+x);
figure,plot(x,y)
axis tight,xlabel('f'),ylabel('g')
title('intensity transformation')
%循环对矩阵中的每个元素进行变换处理
[m,n]=size(X1);
X2=double(X1);
for i=1:m
   for j=1:n
        g(i,j)=c*log(X2(i,j)+1);
   end
end
figure,imshow(mat2gray(g))
```

c=255/log(256)时的对数变换曲线和处理结果分别如图 8.10 和图 8.11 所示。根据所处理图像的不同情况，可选取不同的 c 值，当然处理所得的结果会有所不同，读者可从多次试验中选择符合应用标准的参数值。

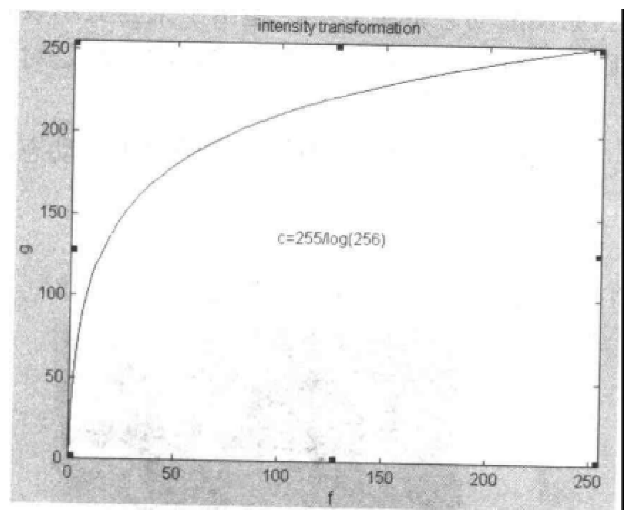


图 8.10　c=255/log(256)对数变换曲线

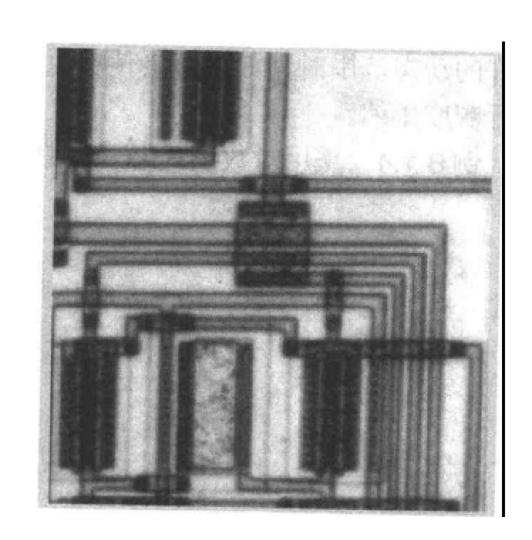


图 8.11　对数变换处理的图像

例 8.1.5　利用 MATLAB 工具箱中的函数 imadjust()变换图像。代码如下：

```
X1=imread('circuit.tif ');
figure,imshow(X1)
J = imadjust(X1,[],[ 0.1 0.8],0.35287)
figure, imshow(J)
```

采用以上参数的变换曲线和处理结果分别如图 8.12 和图 8.13 所示。

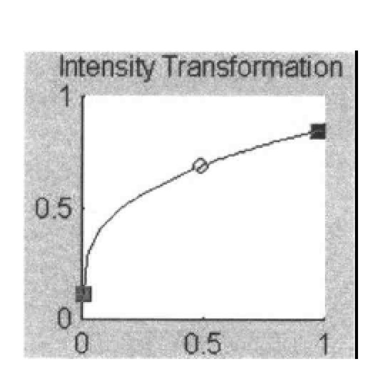


图 8.12 变换曲线

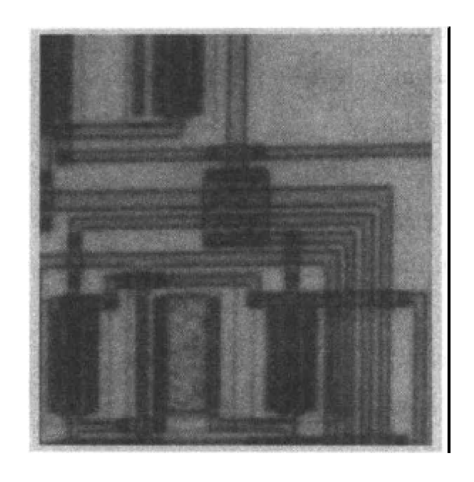


图 8.13 变换的图像

读者同样可以使用不同的参数组合来得到不同的效果。

4. 灰度切分

灰度切分的目的与增强对比度相似，即将某个灰度值范围变得比较突出；所不同的是，这里将所要突出的灰度范围变换成较高的灰度值，而将其余灰度值变换为较低的灰度值。常用的变换曲线如图 8.14 所示。

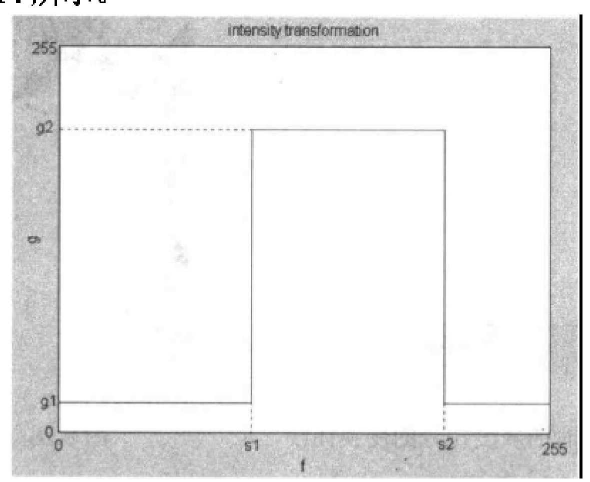


图 8.14 灰度切分的变换曲线

图中的 s1、s2 为用户选定的参数,而 s1 和 s2 之间的灰度范围是用户想要突出的部分。

例 8.1.6 取 s1=100，s2=200，运行如下代码：

```
X1=imread('pout.tif');
figure,imshow(X1)
%绘制变换函数曲线
s1=100;s2=200;g1=20;g2=200;
figure,plot([0,s1,s1,s2,s2,255],[g1,g1,g2,g2,g1,g1])
axis tight,xlabel('f'),ylabel('g')
title('intensity transformation')
[m,n]=size(X1);
```

```
X2=double(X1);
for i=1:m    %循环对矩阵中的每个元素进行变换处理
  for j=1:n
      f=X2(i,j);
      g(i,j)=0;
    if (f>=s1)&(f<=s2)
        g(i,j)=g2;
    else
        g(i,j)=g1;
    end
  end
end
figure,imshow(mat2gray(g))
```

变换曲线和变换结果分别如图 8.15 和图 8.16 所示。

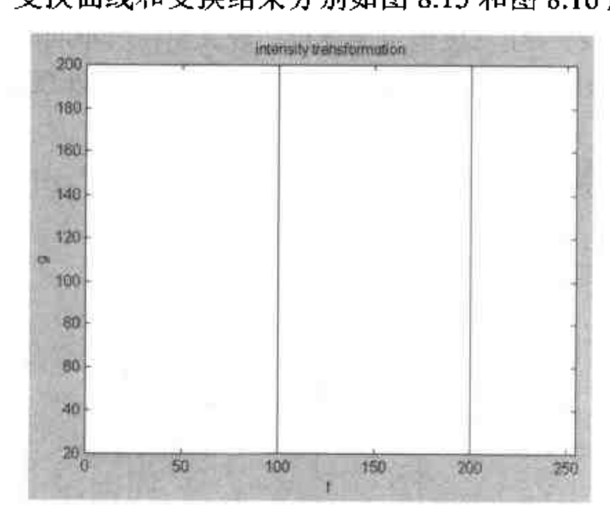


图 8.15 s1=100,s2=200 时的灰度切分的变换曲线



图 8.16 灰度切分的结果

图像二值化可以看作是灰度切分的一种特殊情况，它将灰度值小于某个 f1 的所有像素点都置为 0，而将灰度值大于 f1 的所有像素点都置为 255；如果再将 0 和 255 这两个灰度级分别映射为 0 和 1，就得到了二值图像。

例 8.1.7 取 f1=100，运行如下代码：

```
X1=imread('pout.tif ');
figure,imshow(X1)
%绘制变换函数曲线
f1=100;
figure,plot([0,f1,f1,255],[0,0,255,255])
axis tight,xlabel('f'),ylabel('g')
title('intensity transformation')
%循环对矩阵中的每个元素进行变换处理
```

```
[m,n]=size(X1);
X2=double(X1);
for i=1:m
  for j=1:n
     f=X2(i,j);
     g(i,j)=0;
     if (f>=0)&(f<=f1)
          g(i,j)=0;
     elseif (f>=f1)&(f<=255)
          g(i,j)=255;
     end
  end
end
figure,imshow(mat2gray(g))
```

变换曲线和变换结果分别如图 8.17 和图 8.18 所示。

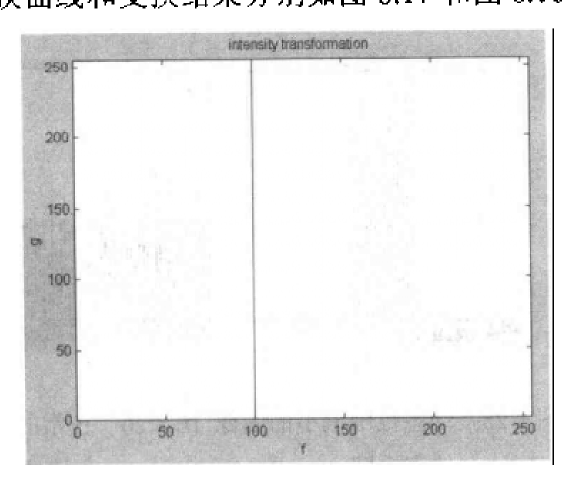


图 8.17 f1=100 时的二值变换曲线



图 8.18 二值化的结果

8.1.2 直方图处理

图像的灰度统计直方图是一个一维离散函数：

$$p(s_k)=\frac{n_k}{n} \qquad k=0,1,\cdots,L-1 \tag{8.3}$$

式中，s_k 是图像 f(x,y) 的第 k 级灰度值；n_k 是 f(x,y) 中具有灰度值 s_k 的像素的个数；n 是图像中像素的总数。由定义式可知，$p(s_k)$ 给出了对 s_k 出现概率的一个估计，所以直方图表明了图像中灰度值的分布情况。因此，可以通过改变直方图的形状来达到增强图像对比度的效果。这种方法是以概率论为基础的，常用的方法有直方图均衡化和直方图规定化。图

8.19～图 8.26 是四幅不同效果的图像以及四幅对应的直方图。通过对比可以看出，偏暗的图像灰度范围很窄，而且主要集中在低灰度级上；偏亮的图像灰度范围主要集中在高灰度级上；灰蒙蒙的图像同样是因为灰度范围窄，动态范围小，主要集中在中间的灰度级上；正常的图像动态范围很大，在各个灰度级上都有像素，从而使图像看起来对比度较大，细节清晰。后面将要提到的直方图均衡化和直方图规定化主要就是基于此种分析，对图像的直方图进行操作，来达到图像增强的目的。

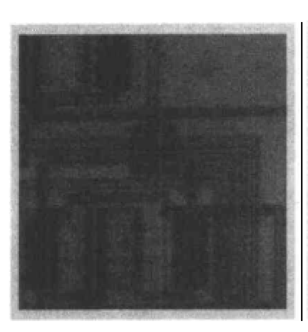


图 8.19 偏暗的图像

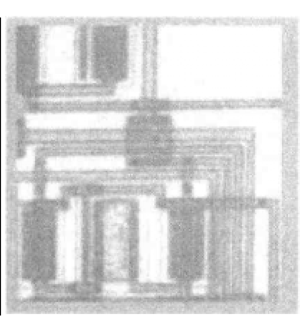


图 8.20 偏亮的图像

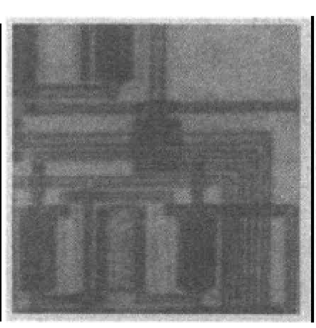


图 8.21 动态范围偏小的图像

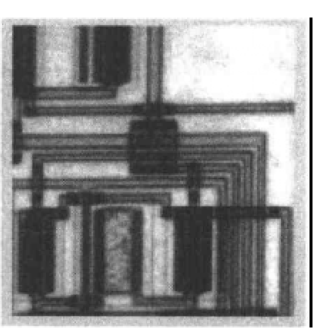


图 8.22 正常的图像

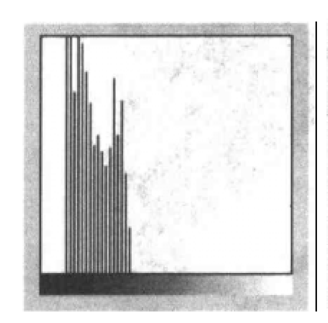


图 8.23 偏暗的图像的直方图

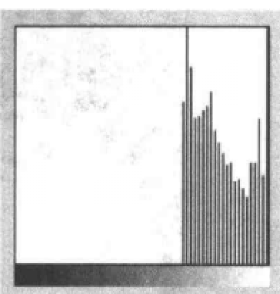


图 8.24 偏亮的图像的直方图

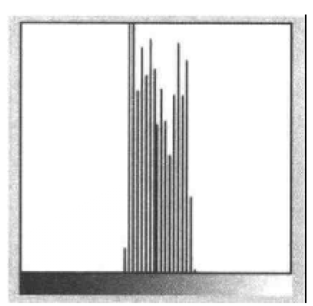


图 8.25 动态范围偏小的图像的直方图

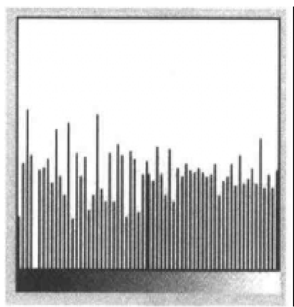


图 8.26 正常的图像的的直方图

1. 直方图均衡化

这个方法的基本思想是把原始图像的直方图变换成均匀分布的形式，这样就增加了像素灰度值的动态范围，从而达到了增强图像整体对比度的效果。具体方法是：

(1) 计算出原始图像的所有灰度级 s_k，k=0，1，…，L-1；

(2) 统计原始图像各灰度级的像素数 n_k；

(3) 用式(8.3)计算原始图像的直方图；

(4) 计算原始图像的累计直方图，即

$$t_k = EH(s_k) = \sum_{i=0}^{k} \frac{n_i}{n} = \sum_{i=0}^{k} p_s(s_i) \qquad 0 \leqslant s_k \leqslant 1;\ k=0,\ 1,\ \cdots,\ L-1 \tag{8.4}$$

(5) 取整计算:

$$t_k = \text{int}\left[(N-1)t_k + \frac{k}{N}\right]$$

(6) 定义映射关系: $s_k \to t_k$;

(7) 统计新直方图各灰度级的像素数 n_k;

(8) 计算新的直方图:

$$p_1(t_k) = \frac{n_k}{n}$$

例 8.1.8 实现直方图均衡化。

这里有一幅图像(如图 8.27 所示),其动态范围较小,而且较暗,反映在直方图上(如图 8.28 所示),就是其直方图所占据的灰度值范围比较窄,而且集中在低灰度值一边。为了使图像更清晰,我们采用直方图均衡化的方法来增加图像灰度动态范围,增强对比度。在 MATLAB 中,可以直接调用 J=histeq (I,n)函数来完成这项工作,其中 I 是原始图像矩阵,J 是变换后所得的图像矩阵。用户可以指定均衡化后的灰度级数 n,缺省值为 64。

实现代码如下:

```
I = imread('tire.tif');
J = histeq(I);
imshow(I)
figure, imshow(J)
figure,imhist(I,64)
figure,imhist(J,64)
```

原始图像、原始直方图、处理后的图像和直方图分别如图 8.27~图 8.30 所示。从处理前后的效果可以看出,许多在原始图像中看不清的细节在直方图均衡化处理所得的图像中都十分清晰。



图 8.27 原始图像

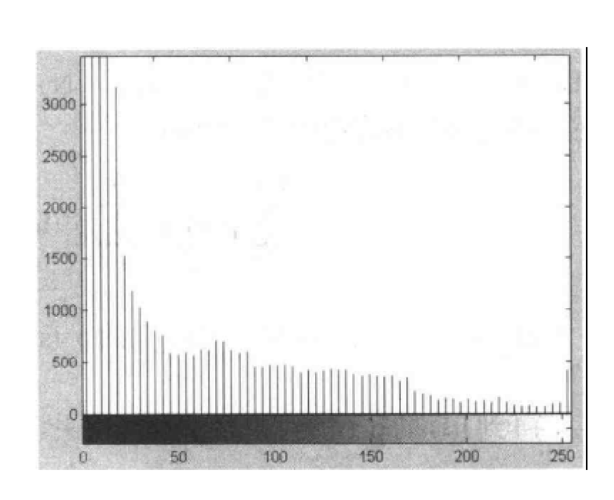


图 8.28 原始直方图



图 8.29 直方图均衡化所得的图像

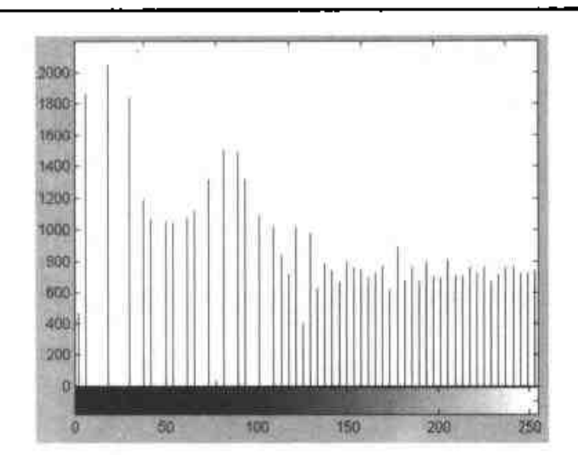


图 8.30 直方图均衡化后的直方图

2. 直方图规定化

直方图均衡化的优点是能自动增强整个图像的对比度，但它的具体增强效果不易控制；处理的结果总是得到全局均衡化的直方图。实际中有时需要变换直方图，使之成为某个特定的形状，从而有选择地增强某个灰度值范围的对比度。这时可采用比较灵活的直方图规定化。直方图规定化方法主要有三个步骤：

(1) 对原始图像的直方图进行灰度均衡化，即

$$t_k = EH_s(s_i) = \sum_{i=0}^{k} p_s(s_i) \qquad k = 0,1,\cdots,M-1 \tag{8.5}$$

(2) 规定所需要的直方图，并计算能使规定的直方图均衡化的变换，即

$$v_l = EH_u(u_j) = \sum_{j=0}^{l} p_u(u_j) \qquad j = 0,1,\cdots,N-1 \tag{8.6}$$

(3) 将第一步所得到的变换翻转过来，也就是将原始图像对应映射到规定的直方图上，将所有的 $p_s(s_i)$ 对应到 $p_u(u_j)$ 中去。对应规则可选用组映射规则(Group Mapping Law，GML)。设有一个整数函数 $I(l)$，$l = 0,1,\cdots,N-1$，满足 $0 \leqslant I(0) \leqslant \cdots \leqslant I(l) \leqslant \cdots \leqslant I(N-1) \leqslant M-1$。现在要确定能使下式达到最小的 $I(l)$。

$$\left|\sum_{i=0}^{I(l)} p_s(s_i) - \sum_{j=0}^{l} p_j(u_j)\right| \qquad l = 0,1,\cdots,N-1 \tag{8.7}$$

如果 $l = 0$，则将其 i 从 0 到 I(0)的 $p_s(s_i)$ 对应到 $p_u(u_0)$ 中去；如果 $l \geqslant 1$，则将其 I 从 $I(l-1)+1$ 到 $I(l)$ 的 $p_s(s_i)$ 对应到 $p_u(u_j)$ 中去。

在 MATLAB 中，调用函数 J = histeq(I,hgram)可以实现直方图规定化，其中 hgram 是由用户指定的向量，规定将原始图像 I 的直方图近似变换成 hgram，hgram 中的每一个元素都

在[0,1]中。

例 8.1.9 实现直方图规定化。

采用图 8.31 所示的规定化函数实现直方图规定化。运行如下代码：

```
I = imread('tire.tif');
hgram=0:255
J = histeq(I,hgram);
imshow(I)
figure, imshow(J)
figure,imhist(I,64)
figure,imhist(J,64)
```

直方图规定化所得的图像及直方图分别如图 8.32 和图 8.33 所示。比较图 8.29 和图 8.30 可知，使用直方图均衡化得到的结果在一些较暗的区域有些细节仍不太清楚；采用图 8.31 所示的规定化函数对同一幅图像进行直方图规定化变换，所得的结果比均衡化更亮，对应于均衡化图像中较暗区域的一些细节更清晰。从直方图上看，灰度值高的一边更为密集。

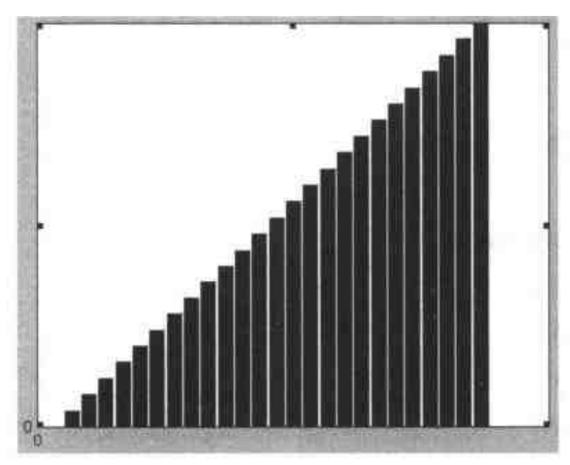


图 8.31 规定化函数



图 8.32 直方图规定化所得的图像

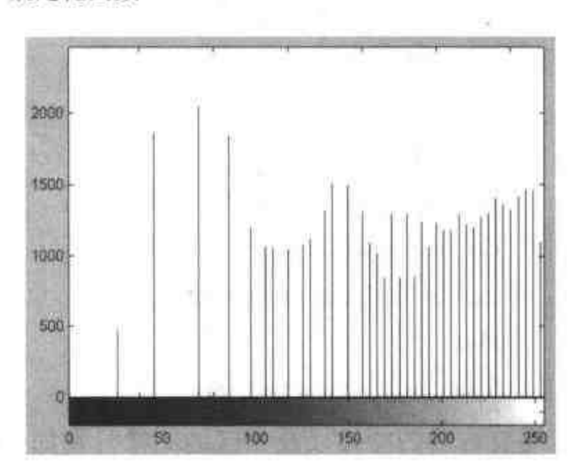


图 8.33 直方图规定化后的直方图

8.1.3 图像间的代数运算

有些图像增强技术是靠对多幅图像进行图像间的代数运算实现的。代数运算是指对两幅图像进行点对点的加、减、乘、除计算而得到输出图像的运算。对于相加和相乘的情形，可能不只有两幅图像参加运算。常用的图像间的代数运算有图像相减和图像相加两种。

1. 图像相减运算

设有图像 $f(x,y)$ 和 $h(x,y)$，它们的差为

$$g(x,y)=f(x,y)-h(x,y)$$

图像相减可以用于去除一幅图像中所不需要的加性图案。加性图案可能是缓慢变化的背景阴影、周期性的噪声或在图像上每个像素处均已知的附加污染等。减法也可以用于检测同一场景的两幅图像之间的变化，例如，通过对一场景的序列图像的减运算可检测运动。在计算用于确定物体边界位置的梯度时，也要用到图像减运算。

例 8.1.10 显示采用离散余弦变换压缩的图像与原图像的差别。

用原图像与压缩后的图像相减，实现代码如下：

```
I = imread('cameraman.tif');
I = double(I)/255;
%计算离散变换矩阵、返回结果为双精度型
T = dctmtx(8);
%实现图像的显示块操作
B = blkproc(I,[8 8],'P1*x*P2',T,T');
mask =[1 1 1 1 0 0 0 0
       1 1 1 0 0 0 0 0
       1 1 0 0 0 0 0 0
       1 0 0 0 0 0 0 0
       0 0 0 0 0 0 0 0
       0 0 0 0 0 0 0 0
       0 0 0 0 0 0 0 0
       0 0 0 0 0 0 0 0];
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);
imshow(I), figure, imshow(I2)
M=I2-I;
figure, imshow(mat2gray(M))
```

压缩前、后的图像以及相减的图像分别如图 8.34、图 8.35 和图 8.36 所示。



图 8.34　原始图像



图 8.35　经压缩—解压后的图像

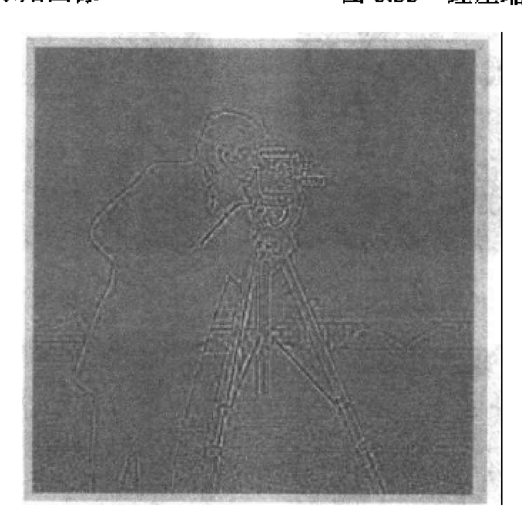


图 8.36　相减所得的图像

2. 图像相加运算

设有图像f(x,y)和h(x,y)，它们的差为

$$g(x,y)=f(x,y)+h(x,y)$$

图像相加的一个重要应用是对同一场景的多幅图像求平均值。这种方法经常用来有效地降低加性随机噪声的影响。图像相加也可以用来将一幅图像的内容叠加到另--幅图像上去，以达到二次曝光的效果。

例 8.1.11　实现图像相加。下面以 saturn.tif 为例，将多幅加入高斯噪声的图像经过相加，求平均，达到去除噪声的目的。实现代码如下：

```
I=imread(saturn.tif);
figure,imshow(I)
[m,n]=size(I);
J(m,n)=0;
for i=1:N
    X=imnoise(I,'gaussian');
    figure,imshow(X)
```

```
    Y=double(X);
    J=J+Y/N;
end
figure,imshow(mat2gray(J))
```

程序中 N 是设定噪声图像的个数。原始图像、三幅随机加入高斯噪声的不同的图像、10 幅噪声图平均得到的结果、100 幅噪声图平均得到的结果分别如图 8.37～图 8.42 所示。

将各个结果比较可知，参加平均的噪声图像越多，去除噪声的效果越好。

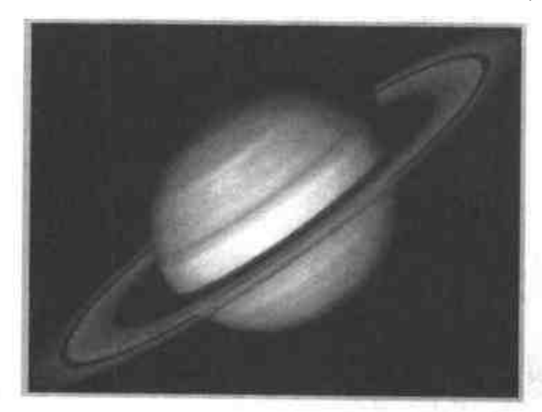


图 8.37 原始 saturn.tif 图像



图 8.38 加噪图像 1

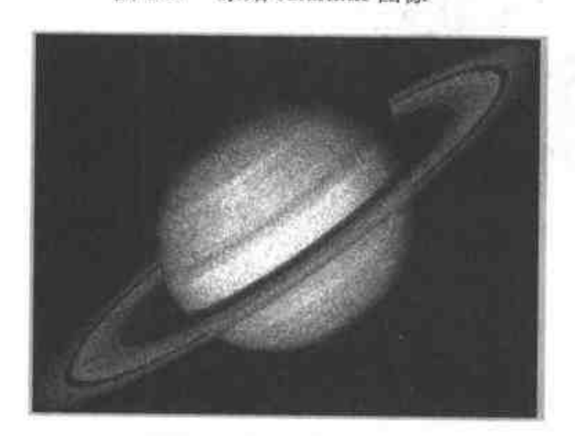


图 8.39 加噪图像 2

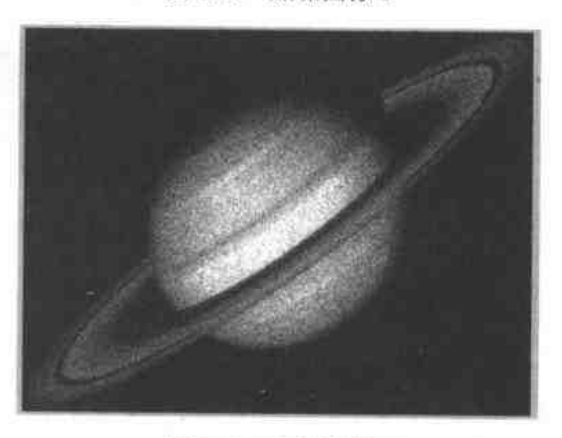


图 8.40 加噪图像 3

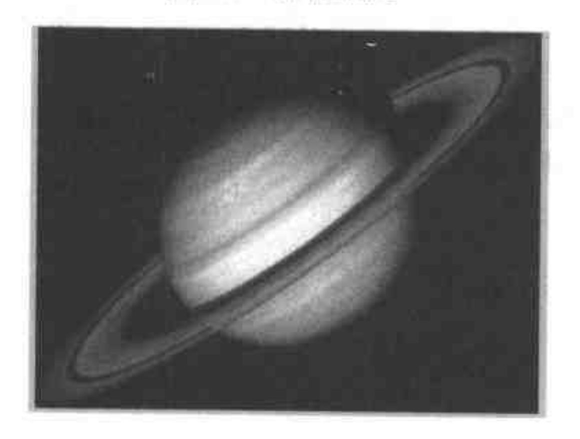


图 8.41 10 幅噪声图平均得到的结果

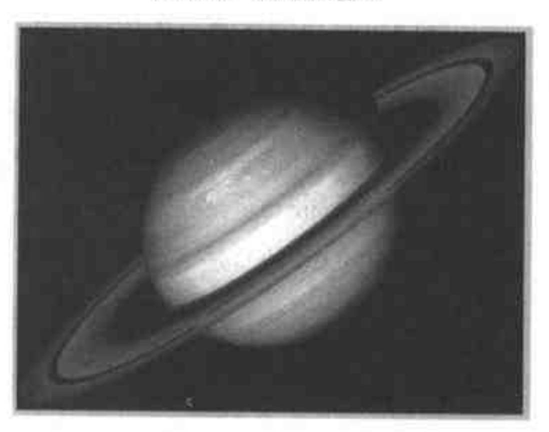


图 8.42 100 幅噪声图平均得到的结果

8.2 空域滤波增强

一般情况下像素的邻域比该像素要大，也就是说，这个像素的邻域中除了本身以外还包括了其它像素。在这种情况下，g(x,y)在(x,y)位置处的值不仅取决于f(x,y)在(x,y)位置处的值，而且取决于f(x,y)在以(x,y)为中心的邻域内所有像素的值。如仍以s和t分别表示f(x,y)和g(x,y)在(x,y)位置处的灰度值，并以n(s)代表f(x,y)在(x,y)邻域内像素的灰度值，则此时式(8.1)可写成

$$t = EH[s, n(s)] \tag{8.8}$$

为在邻域内实现增强操作，常可利用模板与图像进行卷积，每个模板实际上是一个二维数组，其中各个元素的取值确定了模板的功能。这种模板操作也称为空域滤波。

8.2.1 基本原理

空域滤波一般可分为线性滤波和非线性滤波两类。线性滤波器的设计常基于对傅立叶变换的分析，非线性空域滤波器则一般直接对邻域进行操作。另外，各种空域滤波器根据功能又主要分成平滑滤波器和锐化滤波器。平滑可用低通来实现。平滑的目的可分为两类：一类是模糊，目的是在提取较大的目标前去除太小的细节或将目标内的小间断连接起来；另一类是消除噪声。锐化可用高通滤波来实现，锐化的目的是为了增强被模糊的细节。结合这两种分类法，可将空间滤波增强方法分成以下四类：

(1) 线性平滑滤波器(低通)；

(2) 非线性平滑滤波器(低通)；

(3) 线性锐化滤波器(高通)；

(4) 非线性锐化滤波器(高通)。

空域滤波器的工作原理都可借助频域进行分析。它们的基本特点都是让图像在傅立叶空间的某个范围的分量受到抑制，而让其它分量不受影响，从而改变输出图像的频率分布，达到增强的目的。在增强中用到的空间滤波器主要有以下两类：

(1) 平滑(低通)滤波器。它能减弱或消除傅立叶空间的高频分量，但不影响低频分量。因为高频分量对应图像中的区域边缘的灰度值具有较大较快变化的部分，滤波器将这些分量滤去，可使图像平滑。

(2) 锐化(高通)滤波器。它能减弱或消除傅立叶空间的低频分量，但不影响高频分量。因为低频分量对应图像中灰度值缓慢变化的区域，所以与图像的整体特性，如整体对比度和平均灰度值等有关，滤去这些高频分量可使图像锐化。

空域滤波器都是基于模板卷积，其主要工作步骤是：

(1) 将模板在图中漫游，并将模板中心与图中某个像素位置重合；

(2) 将模板上的系数与模板下对应的像素相乘；

(3) 将所有乘积相加；

(4) 将和(模板的输出响应)赋给图中对应模板中心位置的像素。

图8.43(a)中给出了一幅图像的一部分，其中所标的是一些像素的灰度值。现设有一个3

×3 的模板，如图 8.43(b)所示，模板中所标的是模板系数。如将 k_0 所在位置与图灰度值为 s_0 的像素重合(也就是将模板中心放在图中 (x, y) 位置)，模板的输出响应 R 为

$$R = k_0 s_0 + k_1 s_1 + \cdots + k_8 s_8 \tag{8.9}$$

将 R 赋给增强图，作为在 (x, y) 位置的灰度值，如图 8.43(c)所示。如果对原图中每个像素都这样进行，就可得到增强图所有位置的新灰度值。如果我们在设计滤波器时给各个 k 赋不同的值，就可得到不同的高通或低通效果。下面分别介绍在 MATLAB 中应用平滑和锐化滤波器的情况。

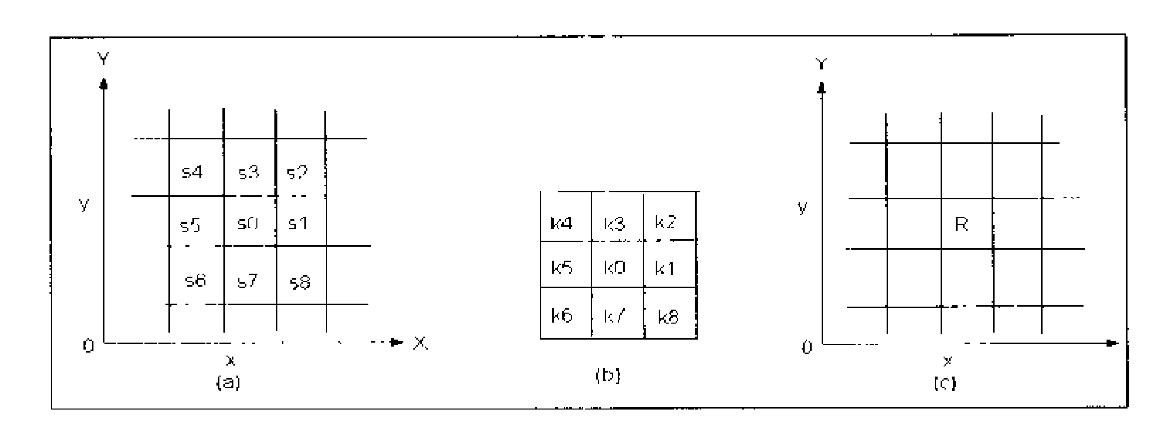


图 8.43　用 3×3 的模板进行空域滤波的示意图

8.2.2　平滑滤波器

1. 线性平滑滤波器

线性低通滤波器是最常用的线性平滑滤波器。这种滤波器的所有系数都是正的。对 3×3 的模板来说，最简单的是取所有系数都为 1。为保证输出图像仍在原来的灰度范围，在计算出 R 后要将其除以 9 再进行赋值。这种方法称为邻域平均法。

例 8.2.1　实现均值过滤器。

以 saturn.tif 为例，加入椒盐噪声，并在 MATLAB 中调用 B = filter2(h,A)实现均值过滤器，实现代码如下：

```
I=imread('eight.tif ');
J=imnoise(I, 'salt &pepper',0.02);
imshow(I)
figure,imshow(J)
K1=filter2(fspecial('average',3),J)/255;
K2=filter2(fspecial('average',5),J)/255;
K3=filter2(fspecial('average',7),J)/255;
figure,imshow(K1)
figure,imshow(K2)
```

```
figure,imshow(K3)
```

其中，B = filter2(h,A)为返回图像 A 经算子 h 滤波的结果；fspecial('average',3)用来创建 3×3 的均值滤波器， fspecial('average',5) 用来创建 5×5 的均值滤波器，fspecial('average',7) 用来创建 7×7 的均值滤波器，其中 3×3 为默认模板。

加入椒盐噪声的图像和均值滤波的图像分别如图 8.44～图 8.47 所示。



图 8.44 加入椒盐噪声的图像



图 8.45 3×3 的均值滤波器处理结果

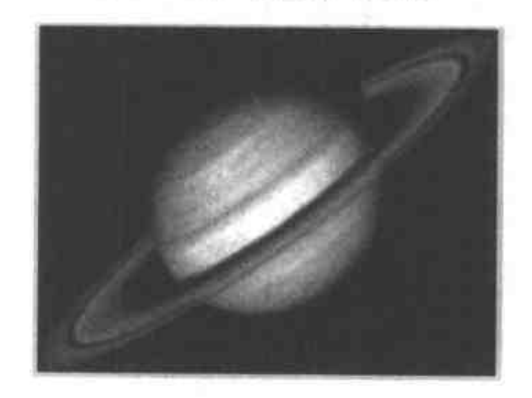


图 8.46 5×5 的均值滤波器处理结果

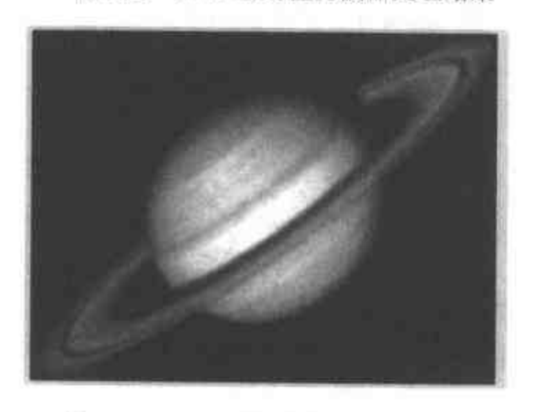


图 8.47 7×7 的均值滤波器处理结果

比较以上采用不同尺寸的均值滤波器进行低通滤波处理的结果可知，当所用的平滑模板的尺寸增大时，消除噪声的效果增强，但同时所得的图像变得更模糊，细节的锐化程度逐步减弱。

此外，维纳滤波器也是经典的线性降噪滤波器。维纳滤波是一种自适应滤波，它能根据图像的局部方差调整滤波器的输出。局部方差越大，滤波器的平滑作用越强。它的最终目标是使恢复图像 $f^*(x,y)$与原始图像 $f(x,y)$的均方误差

$$e^2 = E[(f(x,y) - f^*(x,y))^2]$$

最小。

例 8.2.2 实现维纳滤波器。

在 MATLAB 中调用 J = wiener2(I,[m n],noise)函数，具体代码如下：

```
I = imread('saturn.tif');
J = imnoise(I,' gaussian', 0,0.005);
K1 = wiener2(J, [5,5]);
```

```
figure,imshow(K1)
```

在 J = wiener2(I,[m n],noise)函数中，[m n]指定了滤波器的窗口大小为 m×n，缺省值为 3×3; noise 指定了噪声的功率，默认的是加性噪声(高斯白噪声)。

Wiener2 函数非常适用于图像中的白噪声（比如高斯噪声），加入高斯噪声的图像和维纳滤波的效果分别如图 8.48 和图 8.49 所示。

高斯滤波器在空域中也是具有平滑性能的低通滤波器，可通过调用 h = fspecial('gaussian')函数实现高斯滤波器。

例 8.2.3 实现高斯滤波器，代码如下：

```
I = imread('saturn.tif');
J = imnoise(I,'gaussian', 0,0.005);
figure,imshow(J)
h = fspecial('gaussian');
K=filter2(h,J)/255;
figure,imshow(K)
```

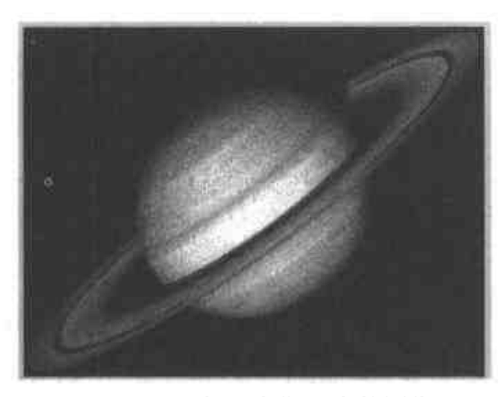


图 8.48 加入高斯噪声的图像

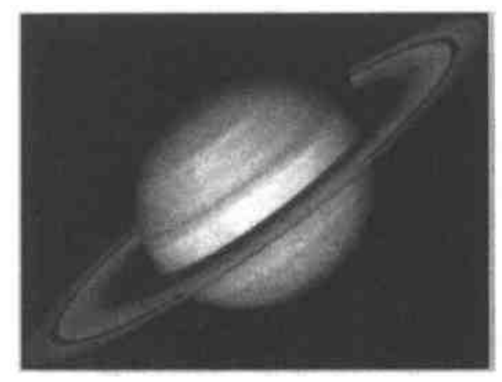


图 8.49 维纳滤波的结果

加入高斯噪声的图像和高斯滤波的结果分别如图 8.50 和图 8.51 所示。



图 8.50 加入高斯噪声的图像



图 8.51 高斯低通滤波的结果

2. 非线性平滑滤波器

中值滤波器是最常用的非线性平滑滤波器。它是一种邻域运算，类似于卷积，但计算的不是加权求和，而是把邻域中的像素按灰度级进行排序，然后选择该组的中间值作为输出像素值。具体步骤是：

(1) 将模板在图像中漫游，并将模板中心与图像中某个像素的位置重合；

(2) 读取模板下各对应像素的灰度值；

(3) 将这些灰度值从小到大排成一列；

(4) 找出这些值里排在中间的一个；

(5) 将这个中间值赋给对应模板中心位置的像素。

由此可以看出，中值滤波器的主要功能就是让与周围像素灰度值的差比较大的像素改取与周围的像素值接近的值，从而可以消除孤立的噪声点。

例 8.2.4 在 MATLAB 中可以调用 B = medfilt2(A,[m n])来实现中值滤波器。

下面仍以 saturn.tif 为例，加入椒盐噪声，并采用中值过滤器，实现代码如下：

```
I = imread('saturn.tif');
J = imnoise(I,'salt & pepper',0.02);
K1 = medfilt2(J, [3,3]);
figure,imshow(K1)
```

其中，函数 K = medfilt2(A,[m n])使用 m × n 的模板来完成中值滤波，默认模板是 3 × 3 的。在以上代码的执行过程中，你会发现中值滤波的计算比卷积的运算要慢，这是由于它需要对邻域中的所有像素按灰度级排序。

采用默认的模板作中值滤波的结果如图 8.52 所示。

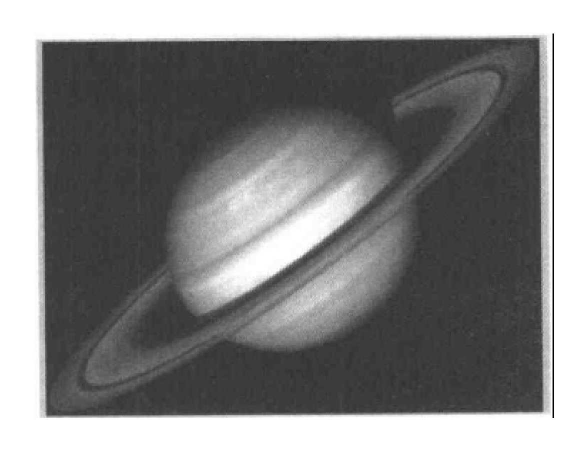


图 8.52 medfilt2(J,[3,3])处理结果

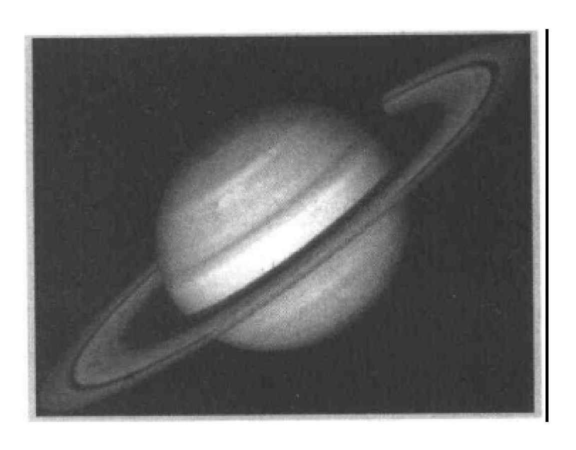


图 8.53 ordfilt2(J,5,ones(3,3))处理结果

例 8.2.5 中值滤波还可以调用 B = ordfilt2(A,order,domain)函数来实现，代码如下：

```
I = imread('saturn.tif');
J = imnoise(I,'salt & pepper',0.02);
K1= ordfilt2(J,5,ones(3,3));
figure,imshow(K1)
```

以上所采用的滤波器也是 3×3 的中值滤波器，效果如图 8.53 所示。

在函数的三个参数中，domain 表示邻域的形状，order 表示所取的像素在邻域像素的排序中所处的位置。当 domain 是全 1 的 3 × 3 方阵时，邻域内共有 9 个元素，排在中间位置的元素为第 5 个。

由以上处理结果可以看出，中值滤波器不像均值滤波器那样，它在衰减噪声的同时不会使图像的边界模糊，这也是中值滤波器受欢迎的主要原因。中值滤波器去噪声的效果依

赖于两个要素：邻域的空间范围和中值计算中所涉及的像素数。一般来说，小于中值滤波器面积一半的亮或暗的物体基本上会被滤掉，而较大的物体则几乎会原封不动地保存下来。因此，中值滤波器的空间尺寸必须根据现有的问题来进行调整。较简单的模板是 N×N 的方形(这里 N 常是奇数)，计算使用到所有 N^2 个像素点。另外，我们也可以使用稀疏分布的模板来节省时间。

例 8.2.6 使用稀疏分布的模板实现中值滤波，代码如下：

```
I = imread('saturn.tif ');
J = imnoise(I,'salt & pepper',0.02);
domain=[0 0 1 0 0;
        0 0 1 0 0;
        1 1 1 1 1;
        0 0 1 0 0;
        0 0 1 0 0];
K1= ordfilt2(J,5,domain);
figure,imshow(K1)
```

效果如图 8.54 所示。

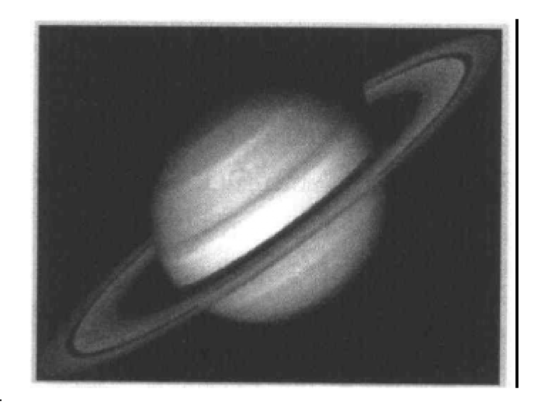


图 8.54 ordfilt2(J,5,domain)处理结果

尽管 domain 是 5×5 的邻域，但由于它是稀疏矩阵，其中非 0 元素只有 9 个，因而在排序完成之后排在中间位置的元素仍为第 5 个。

类似的稀疏矩阵还有很多，比如：

```
domain1=[1 0 0 0 1;
         0 1 0 1 0;
         0 0 1 0 0;
         0 1 0 1 0;
         1 0 0 0 1];
domain2=[0 0 1 0 0;
         0 1 1 1 0;
         1 1 1 1 1;
         0 1 1 1 0;
         0 0 1 0 0];
```

当然，对应于 domain1，中值应取第 5 个像素；而对应于 domain2，中值应取第 7 个像素大的模板；对于其它不同的模板，中值也要依情况而定。计算中值时所使用的像素数的增加跟去噪时的效果之间的关系是非线性的。如果现有的问题需要用到大尺度的中值滤波器，那么用稀疏分布的中值模板或许能得到更令人满意的效果。

中值滤波只是排序统计滤波中的一种。如果邻域中的输入像素已经排好序了，那么中值代表的是第 50 个百分点的数值。其它百分点的数值也可以用。第 0 个百分点可用于最小值滤波器，它用来检测图像中最暗的点；第 100 个百分点可用于最大值滤波器，它用来检

测图像中最亮的点；或利用不是第 50 个百分点来滤波，其结果使图像变暗或变亮。在 MATLAB 中，这几种滤波器都可以用 B = ordfilt2(A,order,domain)函数来实现，分别举例如下：

B = ordfilt2(A,1,ones(3,3))实现 3×3 的最小值滤波器，因为它取全 1 的模板中排在最小位置处的那个像素；

B = ordfilt2(A,9,ones(3,3))实现 3×3 的最大值滤波器，因为它取全 1 的模板中排在最大位置处的那个像素；

B = ordfilt2(A,1,[0 1 0; 1 0 1; 0 1 0])的输出是每个像素的东、南、西、北四个方向相邻像素灰度的最小值，因为它取四邻域的模板中排在最小位置处的那个像素。

8.2.3 锐化滤波器

1. 线性锐化滤波器

线性高通滤波器是最常用的线性锐化滤波器。这种滤波器的中心系数都是正的，而周围的系数都是负的(如果中心和周围正负交换，则产生边缘锐化效果)，所有系数之和为 0。对 3×3 的模板来说，典型的系数取值是：

```
[-1 -1 -1;
 -1  8 -1;
 -1 -1 -1]
```

事实上这是拉普拉斯算子。例如，语句 h=-fspecial('laplacian',0.5)所得到的拉普拉斯算子为：

```
h=
    -0.3333    -0.3333    -0.3333
    -0.3333     2.6667    -0.3333
    -0.3333    -0.3333    -0.3333
```

可以看出这两个模板只是一个比例的不同(经过比例变换后的结果图像是相同的)。当这样的模板放在图像中灰度值为常数或变化很小的区域时，其输出为 0 或很小。这个滤波器将原图像中的零频率分量去除了，也就是将输出图像的平均灰度值变为 0，这样就会有一部分像素灰度值小于 0。在图像处理中，我们一般只考虑正的灰度值，所以还要将输出图像的灰度值范围通过尺度变换变回到所要求的范围。

例 8.2.7 在 MATLAB 中可通过调用 filter2 函数和 fspecial 函数来实现边缘锐化效果，代码如下：

```
I = imread('saturn.tif');
h = fspecial('laplacian');
I2 = filter2(h,I);
imshow(I)
figure,imshow(I2)
```

在 fspecial('laplacian',alpha) 函数中，alpha 控制 Laplacian 算子的形状，取值范围为[0, 1]，缺省值为 0.2。对于一般锐化滤波应当采用 h，因此程序中应当将第二条语句改为 h=-fspecial('laplacian')。请读者自行对比边缘锐化与普通锐化的效果。

图8.55为原始图像，经拉普拉斯算子处理后的效果如图8.56所示。

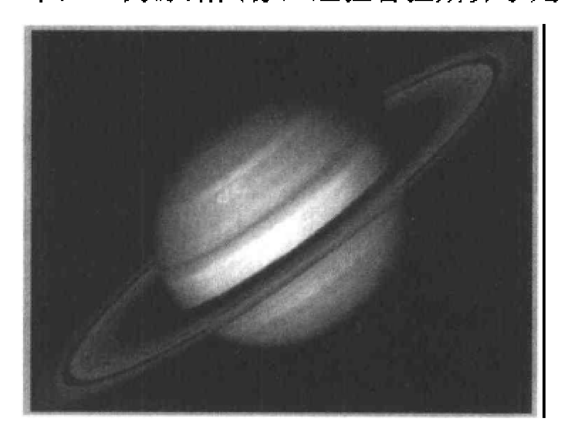



图 8.55　原始图像

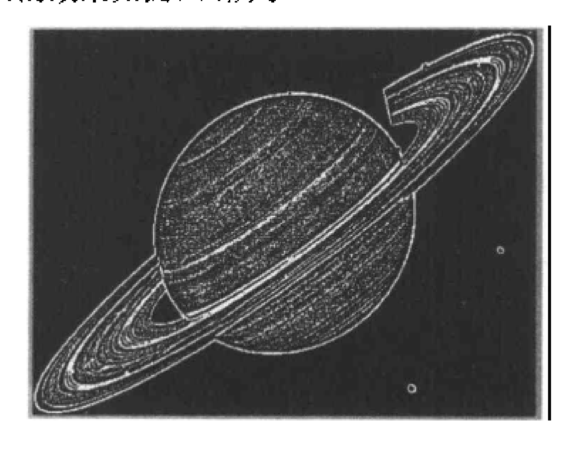


图 8.56　拉普拉斯算子处理的效果

例 8.2.8　采用'unsharp'算子实现对比度增强滤波器，代码如下：

```
I = imread('saturn.tif ');
h = fspecial('unsharp',0.5);
I2 = filter2(h,I)/255;
figurc,imshow(I2)
```

在 h = fspecial('unsharp', alpha)函数中，alpha 控制滤波器的形状，取值范围为[0, 1]，缺省值为 0.2。

处理效果如图8.57所示。关于锐化滤波器，在后面的边缘检测中还要详细介绍，这里为了体系结果的完整，只是简单提到。

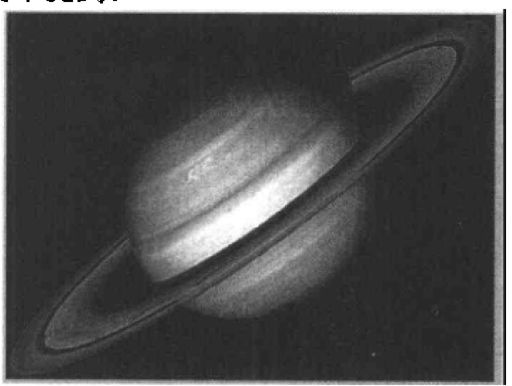


图 8.57　对比度增强滤波器处理的效果

2. 非线性锐化滤波器

邻域平均可以模糊图像，因为平均对应积分，所以利用微分可以锐化图像。图像处理中最常用的微分方法是利用梯度。对一个二维连续函数 f(x,y)，其梯度是一个矢量（需要用两个模板分别沿x和y方向计算），即

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]^{T}$$

其模（以2为模，对应欧氏距离）为

$$\left|\nabla f_{(2)}\right| = mag(\nabla f)\left[\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2\right]^{1/2}$$

在使用中为了计算简便，也可不用上述对应欧氏距离的以 2 为模的方法组合两个模板的输出。一种简单的方法是利用城区距离（以 1 为模），即

$$\left|\nabla f_{(1)}\right| = \left|\frac{\partial f}{\partial x}\right| + \left|\frac{\partial f}{\partial y}\right|$$

另一种简单的方法是利用棋盘距离（以 ∞ 为模），即：

$$\left|\nabla f_{(\infty)}\right| = \max\left\{\left|\frac{\partial f}{\partial x}\right|, \left|\frac{\partial f}{\partial y}\right|\right\}$$

上述这些组合的方法本身都是非线性的。常用的空域微分算子有sobel算子、prewitt算子、高斯—拉普拉斯算子等。

例 8.2.9　对几种边缘增强算子的效果进行比较，代码如下：

```
I1=imread('blood1.tif');
figure,imshow(I1)
h1=fspecial('sobel');
I2=filter2(h1,I1);
figure,imshow(I2)
I3=conv2(I1,h1);
figure,imshow(I3)
h2=fspecial('prewitt');
I4=filter2(h2,I1);
figure,imshow(I4)
h3=fspecial('log');
I5=filter2(h3,I1);
figure,imshow(I5)
```

原始图像如图8.58所示，结果如图8.59～图8.62所示。

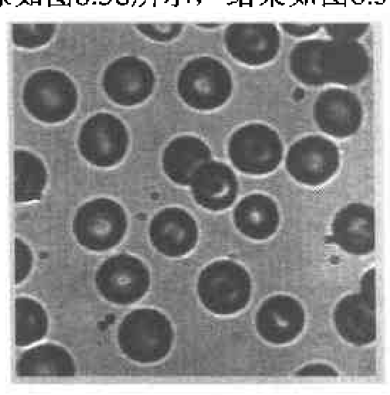


图 8.58　原始图像

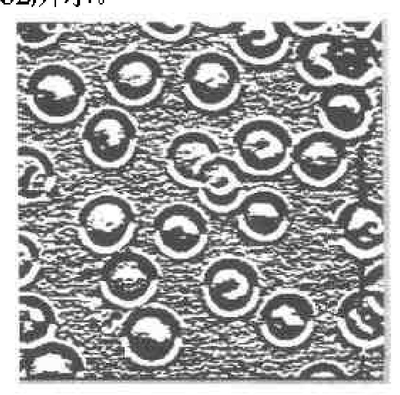


图 8.59　sobel 卷积

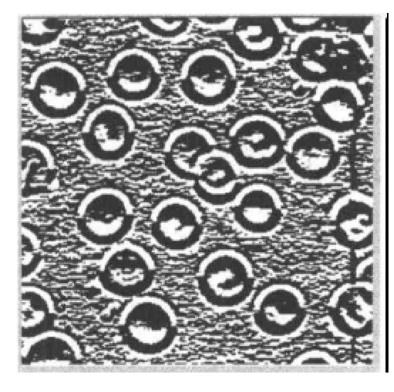




图 8.60 sobel 滤波

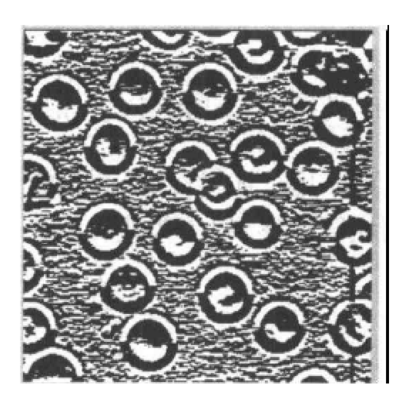


图 8.61 prewitt 滤波

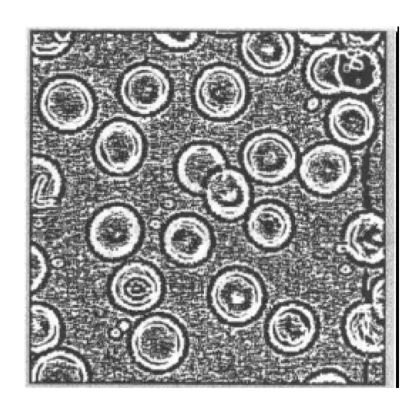


图 8.62 log 滤波

8.3 频域增强

卷积理论是频域技术的基础。设函数 f(x,y) 与线性位不变算子 h(x,y) 的卷积结果是 g(x,y)，即

$$g(x,y)=h(x,y)*f(x,y)$$

那么，根据卷积定理在频域有

$$G(u,v)=H(u,v)F(u,v)$$

其中，G(u,v)、H(u,v)、F(u,v) 分别是 g(x,y)、h(x,y)、f(x,y) 的傅立叶变换。

频域增强的主要步骤是：

(1) 计算所需增强图的傅立叶变换；

(2) 将其与一个(根据需要设计的)转移函数相乘；

(3) 再将结果进行傅立叶反变换以得到增强的图。

频域增强方法有两个关键：

(1) 将图像从空域转换到频域所需的变换(设用 T 表示)以及将图像从频域空间转换回空域所需的变换(设用 T^{-1} 表示)；

(2) 在频域空间对图像进行增强加工的操作(设仍用 EH 表示)，此时对应的增强可表示为

$$g(x,y)=T^{-1}\{EH[T[f(x,y)]]\} \tag{8.10}$$

常用的频域增强方法有低通滤波和高通滤波。以下分别介绍它们在 MATLAB 中是如何实现的。

8.3.1 低通滤波

图像的能量大部分集中在幅度谱的低频和中频部分，而图像的边缘和噪声对应于高频部分。因此能降低高频成分幅度的滤波器就能减弱噪声的影响。

Butterworth 低通滤波器是一种物理上可以实现的低通滤波器。n 阶截断频率为 d0 的 Butterworth 低通滤波器的转移函数为

$$H(u,v)=\frac{1}{1+[d(u,v)/d0]^{2n}}$$

例 8.3.1　实现 Butterworth 低通滤波器，代码如下:

```
I1=imread('Saturn.tif ');
figure,imshow(I1)
I2=imnoise(I1,'salt & pepper');
figure,imshow(I2)
f=double(I2);
%采用傅立叶变换
g=fft2(f);
%数据矩阵平移
g=fftshift(g);
[N1,N2]=size(g);
n=2;
d0=50;
n1=fix(N1/2);
n2=fix(N2/2);
for i=1:N1
  for j=1:N2
       d=sqrt((i-n1)^2+(j-n2)^2);
        %计算Butterworth低通变换函数
        h=1/(1+0.414*(d/d0)^(2*n));
        result(i,j)=h*g(i,j);
  end
end

result =ifftshift(result);
X2=ifft2(result);
X3=uint8(real(X2));
figure,imshow(X3)
```

加噪图像(见图 8.6.3)经处理后如图 8.64 所示。

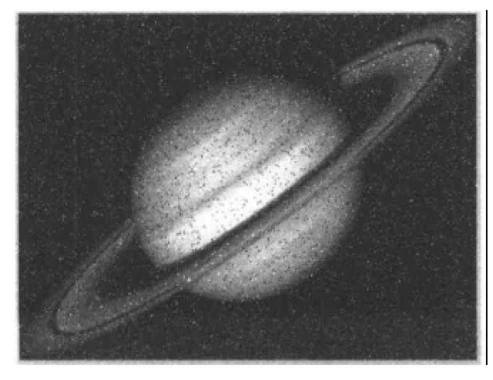


图 8.63　加噪图像

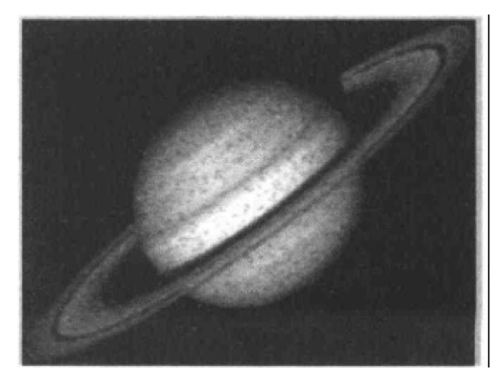


图 8.64　Butterworth 低通滤波器去噪图像

8.3.2 高通滤波

高通滤波也称高频滤波，它的频值在0频率处为单位1，随着频率的增长，传递函数的值逐渐增加；当频率增加到一定值之后，传递函数的值通常又回到0值或者降低到某个大于1的值。在前一种情况下，高频增强滤波器实际上是一种带通滤波器，只不过规定0频率处的增益为单位1。

在实际应用中，为了减少图像中面积大且缓慢变化的成分的对比度，有时让0频率处的增益小于单位1更合适。如果传递函数通过原点，则可以称为laplacian滤波器。

n阶截断频率为d0的Butterworth高通滤波器的转移函数为

$$H(u,v)=\frac{1}{1+[d0/d(u,v)]^{2n}}$$

例8.3.2 实现Butterworth高通滤波器，代码如下：

```
I1=imread('blood1.tif');
figure,imshow(I1)
f=double(I1);
g=fft2(f);                    %采用傅立叶变换
g=fftshift(g);                %数据矩阵平移
[N1,N2]=size(g);
n=2;
d0=5;
n1=fix(N1/2);
n2=fix(N2/2);
for i=1:N1
  for j=1:N2
        d=sqrt((i-n1)^2+(j-n2)^2);
         if d==0
             h=0;
         else
             h=1/(1+(d0/d)^(2*n));
         end
        result(i,j)=h*g(i,j);
  end
end
result =ifftshift(result);
X2=ifft2(result);
X3=uint8(real(X2));
figure,imshow(X3)
```

图8.65为原始图像，处理结果如图8.66所示。

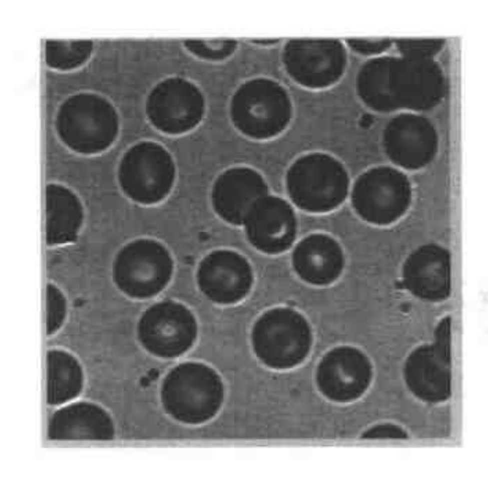


图 8.65 原始图像

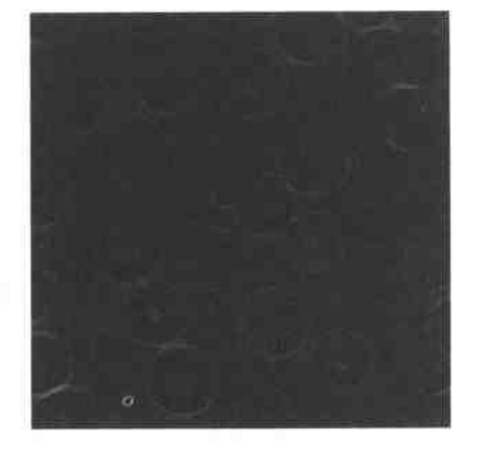


图 8.66 Butterworth 高通滤波图像

从图 8.66 高通滤波图像可见，图像很昏暗，很多细节都看不清了，这是由于图像的大部分能量集中在低频区域，而高通滤波使得图中各区域的边界得到较明显增强的同时滤掉了低频分量，使得图中原来比较平滑区域内部的灰度动态范围被压缩，因而整幅图比较昏暗。

图像处理技术篇

第九章　四叉树分解与边缘检测

本章主要内容：

★　四叉树分解
★　边缘检测

9.1　四叉树分解

四叉树分解是一种基于均匀性检测的图像分割方法，在图像分析和图像压缩中应用广泛。本节介绍四叉树分解的基本原理，应用 MATLAB 图像处理工具箱中的函数来实现四叉树分解。

9.1.1　四叉树分解的基本原理以及 MATLAB 工具箱函数

四叉树分解将原始图像逐步细分成小块，操作的目标是将具有一致性的像素分到同一个小块中。通常这些小块都是方块，只有少数情况分成长方形；因此，当图像是方形且像素点的个数是 2 的整数次幂时，四叉树分解法最适用。这里所说的一致性通常是指方块内部各像素点的灰度值的范围不超过某个阈值，也就是个像素点的灰度值接近程度满足要求。

进行四叉树分解的具体过程是：将方形的原始图像分成四个相同大小的方块，判断每个方块是否满足一致性标准，如果满足，就不再继续分裂；如果不满足，就再细分成四个方块，并对细分得到的方块应用一致性检验。这个迭代重复的过程直到所有的方块都满足一致性标准才停止。最后，四叉树分解的结果可能包含多种不同尺寸的方块。图 9.1 是一幅图像四叉树分解的结果，从中可以看出各种不同尺寸的方块：有的是第二次分裂得到的，尺寸是整幅图的 1/16；有的是第三次分裂得到的，尺寸是整幅图的 1/64，这幅图总共进行了 6 次分裂，才得到最终的结果。

用于四叉树分解的函数有 qtdecomp 函数、qtgetblk 函数和 qtsetblk 函数，以下分别介绍它们的语法及用法。

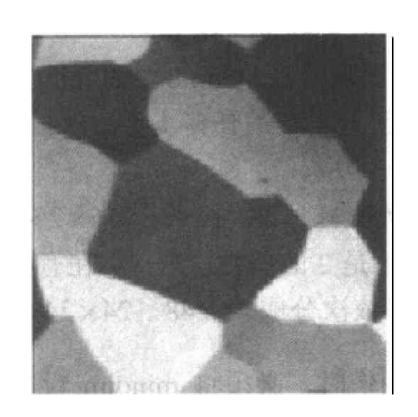
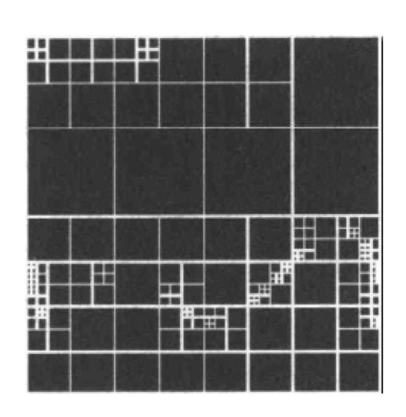



图 9.1　图像的四叉树分解

1. qtdecomp 函数

qtdecomp 函数的语法如下：

S = qtdecomp(I)
S = qtdecomp(I,threshold)
S = qtdecomp(I,threshold,mindim)
S = qtdecomp(I,threshold,[mindim maxdim])
S = qtdecomp(I,fun)
S = qtdecomp(I,fun,P1,P2,...)

其中：

S = qtdecomp(I)表示对灰度图像 I 进行四叉树分解，它递归地将图像分成子块，直到同一子块中的所有元素值都相等。函数将四叉树结构返回到稀疏矩阵 S 中；如果其中的元素 S(k,m) 不等于 0，则 (k,m) 是分解结果中一个方块的左上角，而 S(k,m) 的值就是其所属方块的尺寸。

S = qtdecomp(I,threshold)表示根据阈值 threshold 对原始图像 I 进行四叉树分解。分解过程直到每个方块中最大值元素与最小值元素之间的差值不大于阈值才结束。无论 I 是 uint8 型或 uint16 型，threshold 的取值范围都是 [0,1]；当 I 是 uint8 型时，要将阈值乘以 255 作为实际使用的值；当 I 是 uint16 型时，要将阈值乘以 65 535 作为实际使用的值。

S = qtdecomp(I,threshold,mindim)表示产生的方块尺寸不小于 mindim，即使最后的结果中有一些方块不满足一致性条件。

S = qtdecomp(I,threshold,[mindim maxdim])表示产生的方块尺寸不小于 mindim，不大于 maxdim。尺寸大于 maxdim 的方块即使已经满足了一致性条件也要继续分解。mindim 和 maxdim 都必须是 2 的整数次幂。

S = qtdecomp(I,fun)表示利用函数 fun 来确定方块是否需要继续分解。qtdecomp 函数对 m×m×k 堆栈中尺寸为 m×m 的所有当前方块调用 fun 函数，其中 k 是 m×m 方块的个数。fun 函数返回 k 维向量，每个元素只取 0 或 1。其中 1 表示对应的方块需要继续分解，0 表示对应的方块不需要继续分解，例如，如果 k(3) = 0，那么第三个 m×m 方块不需要继续分解。Fun 是一个字符串，包含函数名、表达式或联机函数。

S = qtdecomp(I,fun,P1,P2,...)表示将 P1,P2,...作为参数传递给 fun。

数据类型说明：在不包含函数的语法中，输入图像可以是 uint8 型的、uint16 型的或 double 型的。在包含函数的语法中，输入图像可以是该函数支持的任何类型。输出矩阵总是 sparse 型的。

当 qtdecomp 函数处理的图像尺寸是 2 的整数次幂（128×128 或 256×256）时，这些图像最小可以分成1×1 的方块。当所处理的图像尺寸不是 2 的整数次幂时，在有些点上方块就不能继续再分解。例如，如果图像是 96×96 的，则可依次分成 48×48、24×24、12×12、6×6、3×3，已经不能分成更小的方块了。处理这种图像时，必须将 mindim 设置为 3（或 2^3）；如果用到包含了函数的语法，则在不能再分的方块处函数不许返回 0。

例 9.1.1 假设下面的 I 是一幅灰度图像，取阈值为 0.5，进行四叉树分解。

```
I = [1    1    1    1    2    3    6    6
     1    1    2    1    4    5    6    8
     1    1    1    1    10   15   7    7
     1    1    1    1    20   25   7    7
     20   22   20   22   1    2    3    4
     20   22   22   20   5    6    7    8
     20   22   20   20   9    10   11   12
     22   22   20   20   13   14   15   16];
S = qtdecomp(I, 0.5)
S =
  (1,1)       2
  (3,1)       2
  (5,1)       1
  (6,1)       1
  (7,1)       1
  (8,1)       1
  (5,2)       1
  (6,2)       1
  (7,2)       1
  (8,2)       1
  (1,3)       1
  (2,3)       1
  (3,3)       2
  (5,3)       1
  (6,3)       1
  (7,3)       2
  (1,4)       1
  (2,4)       1
```

```
(5,4)        1
(6,4)        1
(1,5)        1
(2,5)        1
(3,5)        1
(4,5)        1
(5,5)        1
(6,5)        1
(7,5)        1
(8,5)        1
(1,6)        1
(2,6)        1
(3,6)        1
(4,6)        1
(5,6)        1
(6,6)        1
(7,6)        1
(8,6)        1
(1,7)        1
(2,7)        1
(3,7)        2
(5,7)        1
(6,7)        1
(7,7)        1
(8,7)        1
(1,8)        1
(2,8)        1
(5,8)        1
(6,8)        1
(7,8)        1
(8,8)        1

full(S)
ans =
     2     0     1     1     1     1     1     1
     0     0     1     1     1     1     1     1
     2     0     2     0     1     1     2     0
     0     0     0     0     1     1     0     0
     1     1     1     1     1     1     1     1
```

```
1    1    1    1    1    1    1    1
1    1    2    0    1    1    1    1
1    1    0    0    1    1    1    1
```

2. qtgetblk 函数

qtgetblk 函数的语法如下：

```
[vals,r,c] = qtgetblk(I,S,dim)
[vals,idx] = qtgetblk(I,S,dim)
```

其中：

[vals,r,c] = qtgetblk(I,S,dim)表示返回图像 I 的四叉树分解中 dim×dim 子块的值，存放在数组 vals 中。S 为由 qtdecomp 函数返回的稀疏矩阵，包含了四叉树结构；vals 是一个 dim×dim×k 维的数组，k 是四叉树分解中 dim×dim 维方块的个数，如果四叉树分解中没有指定尺寸的子块，则返回空矩阵；r、c 分别存放行、列坐标的向量，对应着各个子块的左上角。

[vals,idx] = qtgetblk(I,S,dim)表示返回各个子块的左上角的线性索引，存放在向量 idx 中。

数据类型要求：I 可以是 uint8 型、uint16 型或 double 型的；S 是 sparse 型的。

数组 vals 中方块的顺序对应着 I 中方块的列顺序。例如，如果 vals 是 4×4×2 的，vals(:,:,1) 包含的值就来自于 I 中的第一个 4×4 方块，vals(:,:,2) 包含的值就来自于 I 中的第二个 4×4 方块。

例 9.1.2 本例是上面 qtdecomp 函数例子的继续，代码如下：

```
[vals,r,c] = qtgetblk(I,S,2)
vals(:,:,1) =
             1     1
             1     1
vals(:,:,2) =
             1     1
             1     1
vals(:,:,3) =
             1     1
             1     1
vals(:,:,4) =
            20    20
            20    20
vals(:,:,5) =
             7     7
             7     7
r =
    1
    3
    3
```

```
    7
    3
c =
    1
    1
    3
    3
    7
```

3. qtsetblk 函数

qtsetblk 函数的语法如下：

J = qtsetblk(I,S,dim,vals)

其中，J = qtsetblk(I,S,dim,vals)表示将图像 I 的四叉树分解中每个 dim×dim 维子块的值全部替换为 vals 中 dim×dim 维的方块。S 为由 qtdecomp 函数返回的稀疏矩阵，包含了四叉树结构；vals 是一个 dim×dim×k 维的数组，k 是四叉树分解中 dim×dim 维方块的个数。

数据类型要求：I 可以是 uint8 型、uint16 型或 double 型的，S 是 sparse 型的。

数组 vals 中方块的顺序必须与 I 中方块的列顺序相对应。例如，如果 vals 是 4×4×2 的，vals(:,:,1) 包含的值就用来替换 I 中的第一个 4×4 方块，vals(:,:,2) 包含的值就用来替换 I 中的第二个 4×4 方块。

例 9.1.3　本例是上面 qtgetblk 函数例子的继续，代码如下：

```
newvals= cat(3,zeros(2),ones(2),2*ones(2),3*ones(2),4*ones(2))
newvals(:,:,1) =
          0     0
          0     0
newvals(:,:,2) =
          1     1
          1     1
newvals(:,:,3) =
          2     2
          2     2
newvals(:,:,4) =
          3     3
          3     3
newvals(:,:,5) =
          4     4
          4     4
J = qtsetblk(I,S,2,newvals)
J =
   0     0     1     1     2     3     6     6
   0     0     2     1     4     5     6     8
```

```
1    1    2    2    10   15   4    4
1    1    2    2    20   25   4    4
20   22   20   22   1    2    3    4
20   22   22   20   5    6    7    8
20   22   3    3    9    10   11   12
22   22   3    3    13   14   15   16
```

9.1.2 四叉树分解应用

例 9.1.4 以 rice.tif 为例，进行四叉树分解，并显示稀疏矩阵的结果。

首先，显示原始图像，代码如下：

```
I1=imread('rice.tif');
figure,imshow(I1)
```

原始图像如图 9.2 所示。然后，选取阈值为 0.2，对原始图像进行四叉树分解，并以灰度图的形式显示分解所得的稀疏矩阵。

```
S = qtdecomp(I1,0.2);
S2=full(S);
figure,imshow(S2)
```

稀疏矩阵如图 9.3 所示。最后，通过查看 S2 来了解四叉树分解的具体结果，代码和计算结果如下：

```
[vals,r,c] = qtgetblk(I1,S2,2);
size(vals)

ans = 2        2        4166

[vals2,r,c] = qtgetblk(I1,S2,4);
size(vals2)

ans = 4        4        1540

[vals3,r,c] = qtgetblk(I1,S2,8);
size(vals3)

ans = 8        8        317
[vals4,r,c] = qtgetblk(I1,S2,16);
size(vals4)

ans = 16       16       12

[vals5,r,c] = qtgetblk(I1,S2,32);
```

```
size(vals5)

ans = 32        32        0

[vals6,r,c] = qtgetblk(I1,S2,1);
size(vals6)

ans = 1         1         872
```

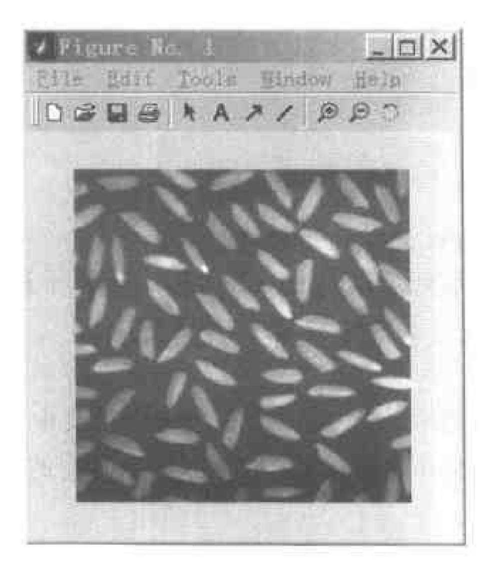




图 9.2　rice.tif 原始图像

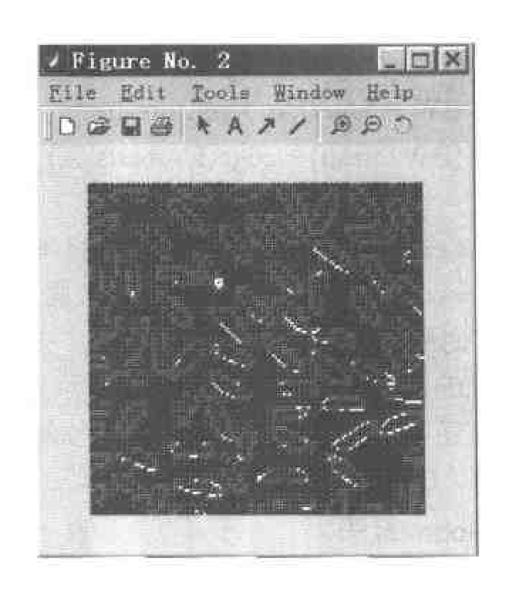


图 9.3　阈值为 0.2 的稀疏矩阵

以上结果说明在 0.2 的阈值下，四叉树分解的最小方块是1×1的，最大方块是32×32维的。当选取的阈值不同时，四叉树分解所得的稀疏矩阵会有差别，读者可以根据应用需要，选取合适的阈值。

9.2　边缘检测

边缘检测在图像处理与计算机视觉中占有特殊的位置，它是底层视觉处理中最重要的环节之一。

9.2.1　边缘检测的基本原理及处理函数

两个具有不同灰度值的相邻区域之间总存在边缘，边缘是灰度值不连续的结果。这种不连续性通常可以利用求导数的方法方便地检测到，一般常用一阶导数和二阶导数来检测。

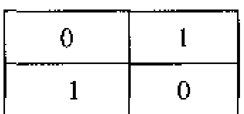
边缘检测的基本思想是首先利用边缘增强算子，突出图像中的局部边缘，然后定义像素的“边缘强度”，通过设置门限的方法提取边缘点集。

常用的边缘检测算子有 Robert 算子、Sobel 算子、Prewitt 算子、LOG 算子和 Canny 算子，以下分别介绍。

1. Robert 算子

Robert 算子是一种利用局部差分算子寻找边缘的算子，其模板如图 9.3 所示。Robert 算子对具有陡峭的低噪声图像效果较好。

1	0
0	1



0	1
1	0

图 9.3　Robert 算子

在 MATLAB 中可以由 edge 函数实现，语法如下：

```
BW = edge(I,'roberts')
BW = edge(I,'roberts',thresh)
[BW,thresh] = edge(I,'roberts',...)
```

其中：

BW = edge(I,'roberts')表示自动选择阈值用 Robert 算子进行边缘检测。

BW = edge(I,'roberts',thresh)表示根据所指定的敏感度阈值 thresh 用 Robert 算子进行边缘检测，它忽略了所有小于阈值的边缘。当 thresh 为空([])时，自动选择阈值。

[BW,thresh] = edge(I,'roberts',...)表示返回阈值。

edge 函数对灰度图像 I 进行边缘检测，返回与 I 同样大的二值图像 BW，其中 1 表示 I 中的边缘，0 表示非边缘。I 是 uint8 型、uint16 型或 double 型的，BW 是 uint8 型的。

2. Sobel 算子

Sobel 算子的两个卷积计算核如图 9.4 所示，图像中的每个点都用这两个核作卷积，第一个核对通常的垂直边缘响应最大，第二个核对水平边缘响应最大。两个卷积的最大值作为该点的输出值，运算结果是一幅边缘幅度图像。Sobel 算子对灰度渐变和噪声较多的图像处理得较好。

1	2	1
0	0	0
1	-2	-1

1	0	-1
2	0	-2
1	0	-1

图 9.4　Sobel 算子

由 edge 函数实现的语法如下：

```
BW = edge(I,'sobel')
BW = edge(I,'sobel',thresh)
BW = edge(I,'sobel',thresh,direction)
[BW,thresh] = edge(I,'sobel',...)
```

其中：

BW = edge(I,'sobel')表示自动选择阈值，用 Sobel 算子进行边缘检测。

BW = edge(I,'sobel',thresh)表示根据所指定的敏感度阈值 thresh，用 Sobel 算子进行边缘检测，它忽略了所有小于阈值的边缘。当 thresh 为空([])时，自动选择阈值。

BW = edge(I,'sobel',thresh,direction)表示根据所指定的敏感度阈值 thresh，在所指定的方

向 direction 上，用 Sobel 算子进行边缘检测。direction 可取的字符串值为'horizontal' (水平方向)、 'vertical'(垂直方向)或'both'(两个方向)。

[BW,thresh] = edge(I,'sobel',...)表示返回阈值。

3. Prewitt 算子

Prewitt 算子的两个卷积计算核如图 9.5 所示，与使用 Sobel 算子的方法一样，图像中的每个点都用这两个核作卷积，取最大值作为输出。Prewitt 算子也产生一幅边缘幅度图像，也是对灰度渐变和噪声较多的图像处理得较好。

1	0	-1
1	0	-1
1	0	-1

1	1	1
0	0	0
-1	-1	1

图 9.5　Prewitt 算子

由 edge 函数实现的语法如下：

BW = edge(I,'prewitt')

BW = edge(I,'prewitt',thresh)

BW = edge(I,'prewitt',thresh,direction)

[BW,thresh] = edge(I,'prewitt',...)

其中：

BW = edge(I,'prewitt')表示自动选择阈值，用 Prewitt 算子进行边缘检测。

BW = edge(I,'prewitt',thresh)表示根据所指定的敏感度阈值 thresh，用 Prewitt 算子进行边缘检测，它忽略了所有小于阈值的边缘。当 thresh 为空([])时，自动选择阈值。

BW = edge(I,'prewitt',thresh,direction)表示根据所指定的敏感度阈值 thresh，在所指定的方向 direction 上，用 Prewitt 算子进行边缘检测。direction 可取的字符串值为'horizontal'(水平方向)、 'vertical'(垂直方向)或'both'(两个方向)。

[BW,thresh] = edge(I,'prewitt',...)表示返回阈值。

4. LOG 算子

拉普拉斯算子的卷积核如图 9.6 所示，它是一个二阶算子，将在边缘处产生一个陡峭的零交叉。拉普拉斯算子是一个线性的、移不变的算子，它的传递函数在频域空间的原点是零，因此经拉普拉斯滤波过的图像具有零平均灰度。LOG 算子先用高斯低通滤波器将图像进行预先平滑，然后用拉普拉斯算子找出图像中的陡峭边缘，最后用零灰度值进行二值化产生闭合的、连通的轮廓，消除了所有内部点。

0	-1	0
-1	4	1
0	-1	0

-1	1	-1
-1	8	-1
1	-1	-1

图 9.6　拉普拉斯算子

由 edge 函数实现的语法如下：

```
BW = edge(I,'log')
BW = edge(I,'log',thresh)
BW = edge(I,'log',thresh,sigma)
[BW,threshold] = edge(I,'log',...)
```

其中：

BW = edge(I,'log')表示自动选择阈值，用 LOG 算子进行边缘检测。

BW = edge(I,'log',thresh)表示根据所指定的敏感度阈值 thresh，用 LOG 算子进行边缘检测，它忽略了所有小于阈值的边缘。当 thresh 为空([])时，自动选择阈值；当指定 thresh 为 0 时，输出图像具有闭合的轮廓，因为其中包含了输入图像中的所有零交叉点。

BW = edge(I,'log',thresh,sigma)表示根据所指定的敏感度阈值 thresh 和标准偏差 sigma，用 LOG 算子进行边缘检测，缺省时 sigma 等于 2，滤波器是 n×n 维的，其中 $n = \mathrm{ceil}(sigma \times 3) \times 2 + 1$

[BW,threshold] = edge(I,'log',...)表示返回阈值。

5. Canny 算子

Canny 算子检测边缘的方法是寻找图像梯度的局部极大值。梯度是用高斯滤波器的导数计算的。Canny 方法使用两个阈值来分别检测强边缘和弱边缘，而且仅当弱边缘与强边缘相连时，弱边缘才会包含在输出中。因此，此方法不容易受噪声的干扰，能够检测到真正的弱边缘。

由 edge 函数实现的语法如下：

```
BW = edge(I,'canny')
BW = edge(I,'canny',thresh)
BW = edge(I,'canny',thresh,sigma)
[BW,threshold] = edge(I,'canny',...)
```

其中：

BW = edge(I,'canny')表示自动选择阈值，用 Canny 算子进行边缘检测。

BW = edge(I,'canny',thresh)表示根据所指定的敏感度阈值 thresh，用 LOG 算子进行边缘检测。thresh 是一个含两个元素的向量，第一个元素是低阈值，第二个元素是高阈值。如果只给 thresh 指定一个值，则此值作为高阈值，而 0.4×thresh 作为低阈值；当 thresh 为空([])时，自动选择低阈值和高阈值。

BW = edge(I,'canny',thresh,sigma)表示根据所指定的敏感度阈值 thresh 和标准偏差 sigma，用 Canny 算子进行边缘检测，缺省时 sigma 等于 1，滤波器的尺寸 sigma 根据要求自动选择。

[BW,threshold] = edge(I,'canny',...)表示返回含两个元素的阈值向量。

6. 零交叉方法

零交叉方法先用指定的滤波器对图像进行滤波，然后寻找零交叉点作为边缘。由 edge 函数实现的语法如下：

```
BW = edge(I,'zerocross',thresh,h)
[BW,thresh] = edge(I,'zerocross',...)
```

其中:

BW = edge(I,'zerocross',thresh,h)表示根据所指定的敏感度阈值 thresh 和滤波器 h，用零交叉方法进行边缘检测。当 thresh 为空([])时，自动选择阈值；当指定 thresh 为 0 时，输出图像具有闭合的轮廓，因为其中包含了输入图像中的所有零交叉点。

[BW,thresh] = edge(I,'zerocross',...)表示返回阈值。

9.2.2　各种边缘检测算子的效果比较

例 9.2.1　分别采用 Robert 算子、Sobel 算子、Prewitt 算子、LOG 算子和 Canny 算子和零交叉方法检测图像'rice.tif '的边缘，代码如下：

```
I = imread('rice.tif ');
BW1 = edge(I,'sobel');
BW2 = edge(I,'roberts');
BW3 = edge(I,'prewitt');
BW4 = edge(I,'log');
BW5 = edge(I,'canny');
h=fspecial('gaussian',5);
BW6 = edge(I,'zerocross',[],h);
subplot(2,3,1),imshow(BW1)
subplot(2,3,2),imshow(BW2)
subplot(2,3,3),imshow(BW3)
subplot(2,3,4),imshow(BW4)
subplot(2,3,5),imshow(BW5)
subplot(2,3,6),imshow(BW6)
```

结果如图9.7所示。

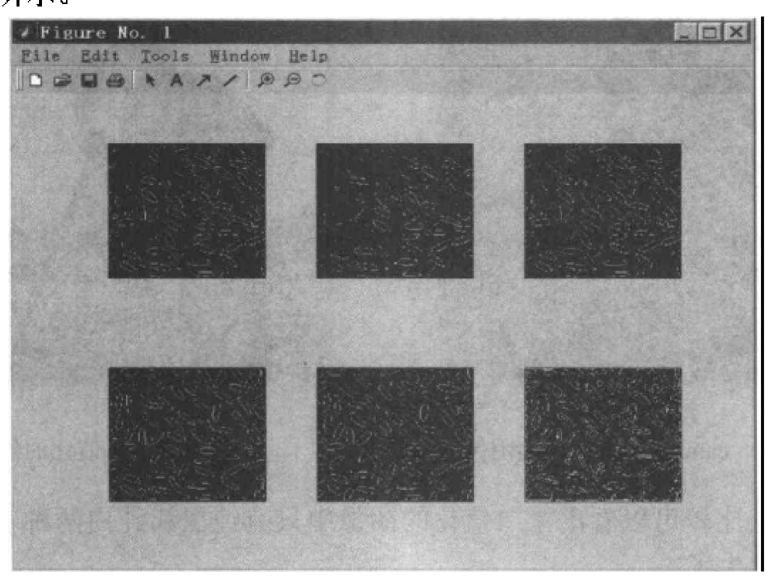


图 9.7　使用各种边缘检测算子得到的边缘图像

第十章 二值图像操作

二值图像中所有的像素只能从 0 和 1 这两个值中取值，因此在 MATLAB 中，二值图像用一个由 0 和 1 组成的二维矩阵表示。这两个可取的值分别对应于关闭和打开，关闭表征该像素处于背景，而打开表征该像素处于前景。以这种方式来操作图像可以更容易识别出图像的结构特征。例如，在二值图像中我们可以非常容易地从图像背景中识别 MATLAB 对象。

二值图像操作只返回与二值图像的形式或结构有关的信息，如果希望对其它类型的图像进行同样的操作，则首先要将其转换为二进制的图像格式，可以通过调用 MATLAB 提供的函数 im2bw 来实现，方法如下：

```
I=imread('cameraman.tif');
figure,imshow(I)
J=im2bw(I);
figure,imshow(J)
```

原图像和二值化的结果分别如图 10.1 和图 10.2 所示。





图 10.1 cameraman.tif 原始图像



图 10.2 二值化的结果

从以上两幅图比较可以看出，二值化的图像中只有纯黑和纯白两种灰度，因而原图像中的很多细节都丢失了。

本章主要内容：

★ 二值形态学基本运算

★ 二值形态学进行图像处理的综合应用

10.1 二值形态学基本运算

二值形态学中的运算对象是集合，也就是二值矩阵，但实际上当涉及两个二值矩阵时并不把它们看作是对等的。通常设 A 为图像矩阵，B 为结构元素矩阵，数学形态学运算是用 B 对 A 进行操作。实际上，结构元素本身也是一个图像矩阵。对每个结构元素矩阵我们指定一个原点，在 MATLAB 中称作中心像素，它是结构元素参与形态学运算的参考点，通常表示用户期望的像素。在 MATLAB 中，中心像素的定义如下：

```
floor((size(SE)+1)/2)
```

其中，SE 是结构要素矩阵。例如：

```
SE =

     0     0     0     0
     0     1     1     1
     0     1     1     1
     0     1     1     1

floor((size(SE)+1)/2)

ans =2      2
```

10.1.1 膨胀

膨胀的算符为⊕，A 用 B 来膨胀写作 $A \oplus B$，这里先将 A 和 B 看作是所有取值为 1 的像素点的集合。其定义为

$$A \oplus B = \left\{x \middle| \left[(\hat{B})_x \cap A\right] \neq \Phi\right\} \tag{10.1}$$

其中 $\hat{B}$ 表示 B 的映像，定义为

$$\hat{B} = \left\{x \middle| x = -b, b \in B\right\} \tag{10.2}$$

$(\hat{B})_x$ 表示对 B 的映像进行位移 x，定义为

$$(M)_x = \left\{y \middle| y = a + x, a \in M\right\} \tag{10.3}$$

式(10.3)表明用 B 膨胀 A 的过程是，先对 B 作关于中心像素的映射，再将其映像平移 x，这里 A 与 B 映像的交集不为空集。换句话说，用 B 来膨胀 A 得到的集合是 $\hat{B}$ 的位移与 A 至

少有一个非零元素相交时 B 的中心像素位置的集合。根据这个解释，(10.1)也可以写成

$$A \oplus B = \left\{ x \middle| \left[(\hat{B})_x \cap A \right] \subseteq A \right\} \tag{10.4}$$

式(10.4)可帮助我们借助卷积的概念来理解膨胀操作。如果将 B 看作是一个卷积模板，膨胀就是先对 B 作关于中心像素的映射，再将映像连续地在 A 上移动而实现的。

下面举例说明膨胀运算的过程。

图 10.3(a)中填入 1 的部分为集合 A；图 10.3(b)中填入 2 的部分为集合 B，标有(2)的位置表示中心像素，这并不代表其所在位置处的像素的灰度值为 2，而只是为了与集合 A 相区分；图 10.3(c)是图 10.3(b)的映像；图 10.3(d)中标有 1 的像素为 A 中原来的位置，标有 2 的像素表示膨胀出来的部分，合起来就是膨胀的结果，也就是$A \oplus B$。从图中可以明显看到，膨胀运算将原图像区域扩大了。

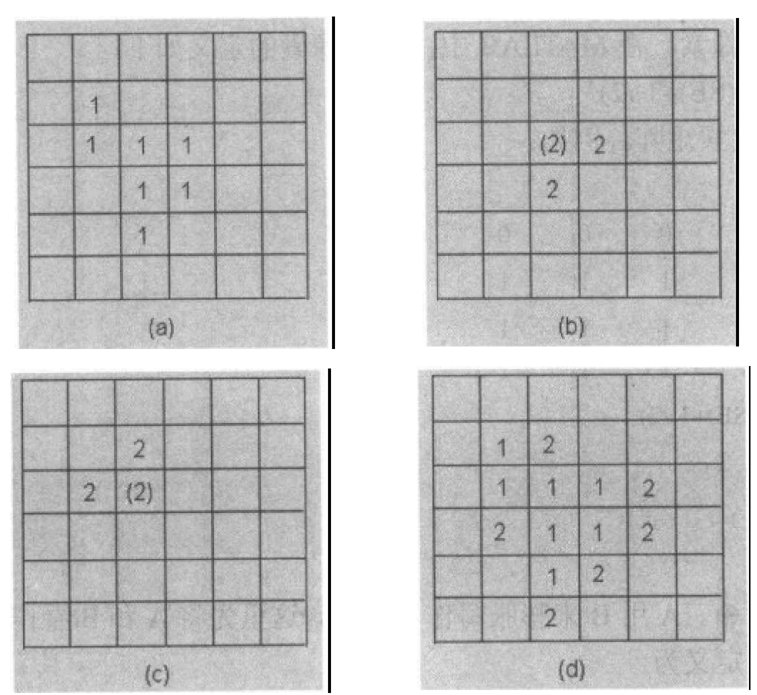


图 10.3　膨胀运算过程示例

例 10.1.1　用 MATLAB 图像处理工作箱中提供的 dilate 函数来实现膨胀操作。采用以下结构要素矩阵：

```
SE = 1    1
     1    1
     1    1
     1    1
     1    1
     1    1
```

对前面得到的 cameraman.tif 的二值化图像进行膨胀操作，代码如下：

```
I=imread('cameraman.tif');
```

```
figure,imshow(I)
J=im2bw(I);
figure,imshow(J)
SE=ones(6,2);
BW1=dilate(J,SE);
figure,imshow(BW1)
```

膨胀的结果如图 10.4 所示。



图 10.4 使用 dilate 函数膨胀的结果

dilate 函数有多种形式：

BW2 = dilate(BW1,SE)

BW2 = dilate(BW1,SE,alg)

BW2 = dilate(BW1,SE,...,n)

其中：

BW2 = dilate(BW1,SE)表示使用二值结构要素矩阵 SE 对图像数据矩阵 BW1 执行膨胀操作。输入图像 BW1 的类型为 double 或 uint8，输出图像 BW2 的类型为 uint8。

BW2 = dilate(BW1,SE,alg)表示使用指定的算法执行膨胀操作，alg 用来指定算法的字符串，可以取的值为'spatial'(在空域内处理图像，这是缺省值)和'frequency'(在频域内处理图像)。这两种算法产生的结果是一样的，所不同的是处理速度和内存的占用。对于很大的图像和结构要素矩阵，频域算法比空域算法要快，但占用的内存要大得多。

BW2 = dilate(BW1,SE,...,n)表示执行膨胀操作 n 次。

值得说明的是，除非你的系统内存很大，否则不要选择频域算法；如果你的内存不足却选用了频域算法，则实际上比空域算法还要慢，因为要占用虚拟内存页。如果频域算法极度地降低了你的系统速度，或者产生了“内存不足”的消息，那么应改用空域算法。

此外，MATLAB 提供了预定义的形态运算函数 bwmorph，其语法如下：

BW2 = bwmorph(BW1,operation)

BW2 = bwmorph(BW1,operation,n)

其中：

BW2 = bwmorph(BW1,operation)表示对图像 BW1 作指定的形态学运算 operation。

BW2 = bwmorph(BW1,operation,n)表示对图像 BW1 作 n 次指定的形态学运算 operation。n 可以是 Inf，在这种情况下，操作将持续到图像不再变化为止。

BW1 可以是 double 型或 uint8 型，BW2 是 uint8 型，operation 可选以下值：

'bothat'：闭包运算，即先腐蚀，再膨胀，然后减去原图。

'erode'：用结构元素ones(3)腐蚀运算。

'shrink'，n = Inf：将对象收缩成点。它消除像素，使没有孔的对象缩成一点；使有孔对象缩成外层边缘，在每个孔之间缩成一个相连的环。此项保持欧拉数。

'bridge'：连接运算，将原来不相连的像素连接起来，例如：

```
1 0 0          1 0 0
1 0 1   变成   1 1 1
0 0 1          0 0 1
```

'fill'：填充孤立的内部像素点(中间是0，周围是1)，类似于以下情况：

1 1 1
2 0 1
1 1 1

'skel'，n = Inf：消除对象边缘上的点，但不使对象分解；剩下的像素构成图像的骨架。此项保持欧拉数。

'clean'：清除孤立的亮点(中间是1，而周围都是0)，类似于以下的情况：

0 0 0
0 1 0
0 0 0

'hbreak'：断开H型联接的像素，例如：

1 1 1		1 1 1
0 1 0	变成	0 0 0
1 1 1		1 1 1

'spur'：去掉小短枝像素，例如：

0 0 0 0		0 0 0 0
0 0 0 0		0 0 0 0
0 0 1 0	变成	0 0 0 0
0 1 0 0		0 1 0 0
1 1 0 0		1 1 0 0

'close'：执行二值闭运算(先膨胀，再扩张)。

'majority'：若像素的 8 邻域中有 5 个以上的像素为 1，则像素值为 1，否则为 0。

'thicken'，n = Inf：在对象的外部不断增加像素，但要保证所增加的像素不会导致原来不连接的对象成为 8-连接；此项保持欧拉数。

'diag'：使用对角线填充来消除背景的 8-连接，例如：

0 1 0		0 1 0
1 0 0	变成	1 1 0
0 0 0		0 0 0

'open'：开启运算(先腐蚀，再膨胀)。

'thin'，n = Inf：将对象细化成线。如果是无孔的对象，则缩成最小连接的一笔；如果是有孔的对象，则缩成外层边缘，而在每个孔之间缩成相连的环。此项保持欧拉数；

'dilate'：用结构元素 ones(3)膨胀。

'remove'：去掉内部像素点，即若像素的四邻域都为 1，则像素值为 0，使得边缘的像素是亮的。

'tophat'：用原图减去开启运算后的图。

以上这些不同的功能将在以后分别介绍。

例 10.1.2　利用 bwmorph 函数来实现膨胀，代码如下：

```
I=imread('cameraman.tif ');
figure,imshow(I)
J=im2bw(I);
```

```
figure,imshow(J)
BW1= bwmorph(J, 'dilate');
figure,imshow(BW1)
```

效果如图 10.5 所示。


图 10.5 使用 bwmorph 函数膨胀的结果

虽然利用 BW1= bwmorph(J, 'erode')函数时，腐蚀的结构要素矩阵是ones(3)：

```
ans =  1  1  1
       1  1  1
       1  1  1
```

但与前面利用 dilate 函数膨胀的结果相比，差别不大，原因在于所采用的结构要素矩阵相对于整个图像矩阵来说很小，因而两种情况下作用的结果很相似。

10.1.2 腐蚀

腐蚀的算符为Θ，A 用 B 来腐蚀写作 AΘB 。其定义为

$$A\Theta B=\left\{x\middle|(B)_x\subseteq A\right\} \tag{10.5}$$

式(10.5)表明 A 用 B 来腐蚀的结果是所有 x 的集合，其中 B 平移 x 后仍在 A 中。换句话说，用 B 来腐蚀 A 得到的集合是 B 完全包括在 A 中时 B 的中心像素位置的集合。

下面举例说明腐蚀运算的过程。

图 10.6(a)中填入 1 的部分为集合 A；图 10.6(b)中填入 2 的部分为集合 B，标有(2)的位置表示中心像素，这并不代表其所在位置处的像素的灰度值为 2，而只是为了与集合 A 相区分；图 10.6(c)中标有 1 的像素表示腐蚀后 A 中保留下来的像素，标有 2 的像素表示 A 中被腐蚀掉的像素。从图中很明显可以看到腐蚀运算将原图像区域缩小了。

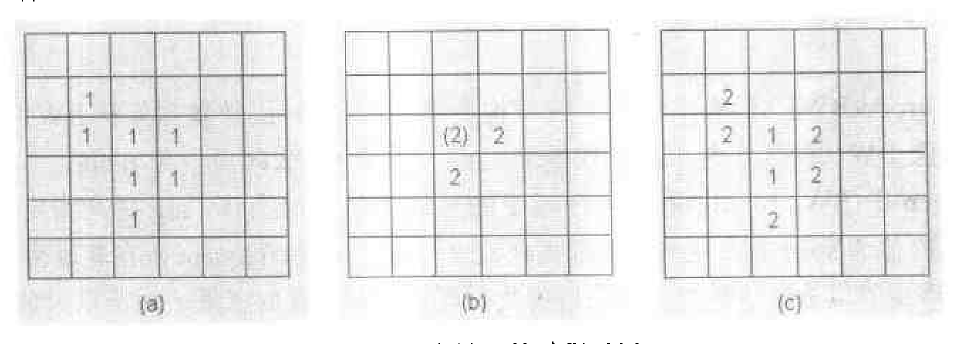


图 10.6 腐蚀运算过程示例

例 10.1.3 用 MATLAB 图像处理工作箱中提供的 erode 函数来实现腐蚀操作。采用以下结构要素矩阵：

```
SE = 1  0  0  0  0
     0  1  0  0  0
     0  0  1  0  0
     0  0  0  1  0
     0  0  0  0  1
```

对前面得到的 cameraman.tif 的二值化图像进行腐蚀操作，代码如下：

```
I=imread('cameraman.tif');
figure,imshow(I)
J=im2bw(I);
figure,imshow(J)
SE=eye(5);
BW1=erode(J,SE);
figure,imshow(BW1)
```

腐蚀的结果如图 10.7 所示。



图 10.7 使用 erode 函数腐蚀的结果

erode 函数也有几种形式：

```
BW2 = erode(BW1,SE)
BW2 = erode(BW1,SE,alg)
BW2 = erode(BW1,SE,...,n)
```

其中：

BW2 = erode(BW1,SE)表示使用二值结构要素矩阵 SE 对图像数据矩阵 BW1 执行腐蚀操作。输入图像 BW1 的类型为 double 或 uint8，输出图像 BW2 的类型为 uint8。

BW2 = erode(BW1,SE,alg)表示使用指定的算法执行腐蚀操作，alg 用来指定算法的字符串，可以取的值为'spatial'(在空域内处理图像，这是缺省值)和'frequency'(在频域内处理图像)。这两种算法产生的结果是一样的，所不同的是处理速度和内存的占用。对于很大的图像和结构要素矩阵，频域算法比空域算法要快，但占用的内存要大得多。

BW2 = erode(BW1,SE,...,n)表示执行腐蚀操作 n 次。

值得说明的是，除非你的系统内存很大，否则不要选择频域算法；如果你的内存不足却选用了频域算法，则实际上比空域算法还要慢，因为要占用虚拟内存页。如果频域算法极度地降低了你的系统速度，或者产生了“内存不足”的消息，那么应改用空域算法。

例 10.1.4 利用 bwmorph 函数来实现腐蚀，代码如下：

```
I=imread('cameraman.tif');
figure,imshow(I)
```

```
J=im2bw(I);
figure,imshow(J)
BW1= bwmorph(J,'erode');
figure,imshow(BW1)
```

效果如图 10.8 所示。

虽然利用 BW1= bwmorph(J, 'erode')函数时，腐蚀的模板是 ones(3)：

```
ans =  1    1    1
       1    1    1
       1    1    1
```

但与前面利用 erode 函数腐蚀的结果相比，只有细小的差别，原因在于所采用的结构要素矩阵相对于整个图像矩阵来说很小，因而两种情况下作用的结果很相似。



图 10.8　使用 bwmorph 函数腐蚀的结果

10.1.3　膨胀与腐蚀的对偶性

膨胀与腐蚀这两种运算是紧密联系在一起的，一种运算对目标的操作相当于另一种运算对图像背景的操作。膨胀与腐蚀的对偶性可表示为

$$(A \oplus B)^c = A^c \Theta \hat{B} \tag{10.6}$$

$$(A \Theta B)^c = A^c \oplus B \tag{10.7}$$

下面图解说明膨胀与腐蚀的对偶性。

图 10.9 中，图(a)中填入 1 的部分为集合 A；图(b)中填入 2 的部分为集合 B，标有(2)的位置表示中心像素，这并不代表其所在位置处的像素的灰度值为 2，而只是为了与集合 A 相区分；图(c)是腐蚀的结果，标有 1 的像素表示腐蚀后 A 中保留下来的像素，标有 2 的像素表示 A 中被腐蚀掉的像素；图(d)中标有 1 的像素为 A 中原来的位置，标有 2 的像素表示膨胀出来的部分，合起来就是膨胀的结果；图(e)中值为 1 的所有像素构成了 A 的补集；图(f)是 B 的映像；图(g)是 $A^c \Theta \hat{B}$ 的结果，也就是图(f)腐蚀图(e)的结果，其中值为 1 的所有像

素是原来图(e)中未被腐蚀掉的像素，而值为 2 的像素是原来图(e)中被腐蚀掉的像素；图(h)是 $A^c \oplus \hat{B}$ 的结果，也就是由图(f)膨胀图(e)的结果，其中值为 1 的所有像素正是原来图(e)中的像素，而值为 2 的像素是膨胀出来的。比较图(d)和图(g)可以验证式 10.6，比较图(c)和图(h)可以验证式 10.7。

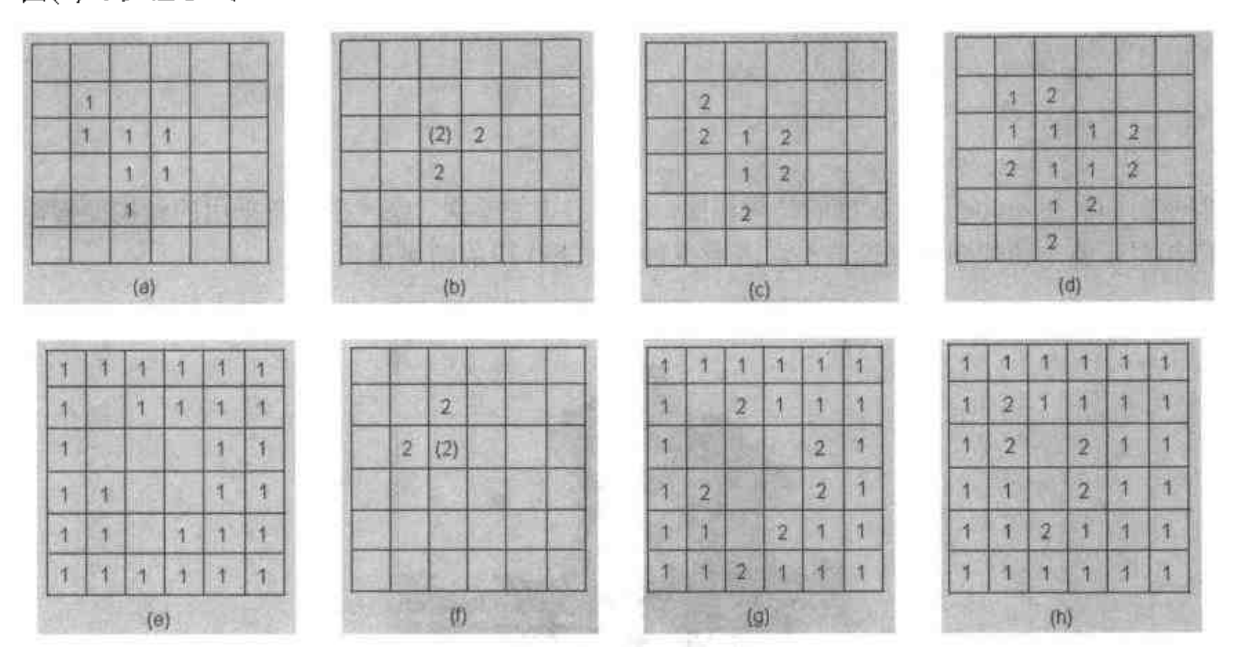


图 10.9 膨胀与腐蚀的对偶性图解

10.1.4 开启和闭合

由于膨胀和腐蚀并不是互为逆运算，因而可以将它们级连结合使用。开启就是先对图像进行腐蚀，然后膨胀其结果。闭合就是先对图像进行膨胀，然后腐蚀其结果。它们也是数学形态学中的重要运算。

开启的算符为∘，A 用 B 来开启写作 $A \circ B$，其定义为

$$A \circ B = (A \Theta B) \oplus B \tag{10.8}$$

闭合的算符为•，A 用 B 来闭合写作 $A \bullet B$，其定义为

$$A \bullet B = (A \oplus B) \Theta B \tag{10.9}$$

开启和闭合这两种运算都可以除去比结构元素小的特定图像细节，同时保证不产生全局几何失真。开启运算可以把比结构元素小的突刺滤掉，切断细长搭接而起到分离作用。闭合运算可以把比结构元素小的缺口或孔填充上，搭接短的间断而起到连通作用。

开启和闭合可以从图像中提取与其结构元素相匹配的形状，由以下开启特性定理和闭合特性定理可以得到

$$A \circ B = \left\{ x \in A \middle| 对某些 t \in A \Theta B, x \in (B)_t 和 (B)_t \subseteq A \right\} \tag{10.10}$$

$$A \bullet B \left\{ x \middle| x \in (\hat{B})_t \Rightarrow (\hat{B})_t \cap A \neq \Phi \right\} \tag{10.11}$$

式(10.10)表明，用B开启A就是选出了A中的某些与B相匹配的点，这些点可由完全包含在A中的结构元素B的平移得到。式(10.11)表明，用B对闭合A的结果包括所有满足如下条件的点，即该点可被映射和位移的结构元素覆盖时，A与经过映射和位移的结构元素的交集不为0。

例 10.1.5　结合 dilate 函数和 erode 函数来实现开启。下面以 circbw.tif 图为例展示开启的效果(见图 10.10～图 10.13)，采用一个40×30的全1的结构要素矩阵，闭合操作的实现代码如下：

```
I=imread('circbw.tif');
figure,imshow(I)
SE=ones(40,30);
BW1=erode(I,SE);
figure,imshow(BW1)
BW2=dilate(BW1,SE);
figure,imshow(BW2)
```

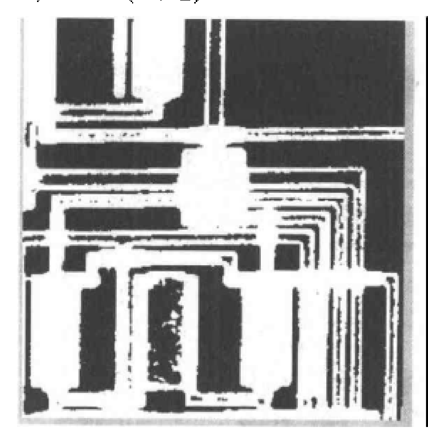


图 10.10　circbw.tif 原始图像

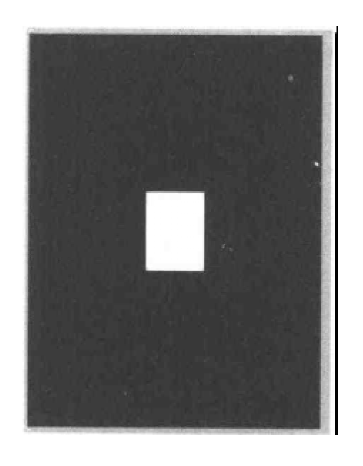


图 10.11　结构要素矩阵

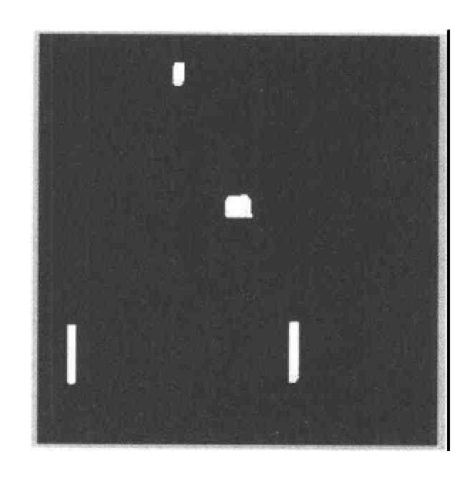


图 10.12　开启的腐蚀结果

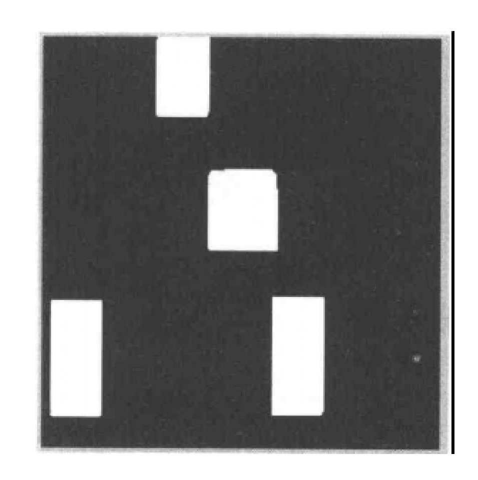


图 10.13　开启的最终结果

从以上四幅图可以看出，原始图像中有很多比结构要素矩阵小的细小轮廓，在开启操作时都将原始图像中的细节去掉了，而将其中较大的方形轮廓按照原来的大小显示出来了。

例 10.1.6 利用 bwmorph 函数来实现开启，代码如下：

```
I=imread('circbw.tif');
figure,imshow(I)
BW1= bwmorph(I, 'open');
figure,imshow(BW1)
```

效果如图 10.14 所示，与前面利用 erode 函数和 dilate 函数的结果相比差别很大，造成这种结果的主要原因在结构要素矩阵。调用 BW1= bwmorph(I, 'open')函数时的结构要素矩阵(见图 10.14)，同图 10.11 的结构要素矩阵相比要小很多，而在 circbw.tif 原始图像中比图 10.14 中的结构要素小的图像细节很少，所以操作的结果就是只有少数细小的连接处断开了，而不像图 10.13 那样效果明显。因此，如果想要达到特定的目的，最好采用原始的方法实现开启操作，也就是先腐蚀，然后膨胀。采用这种方法时结构要素矩阵是可由用户根据需要随意设定的，会比较理想。利用 bwmorph 函数开启的结果如图 10.15 所示。

闭合操作也是同样的道理，采用原始的方法，也就是先膨胀，然后腐蚀，这种方法可由用户控制结构要素，根据需要进行调节，所以比调用 BW1= bwmorph(I, 'close')函数时处理的效果更加理想。

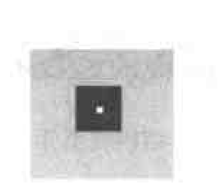


图 10.14 ones(3)结构要素矩阵

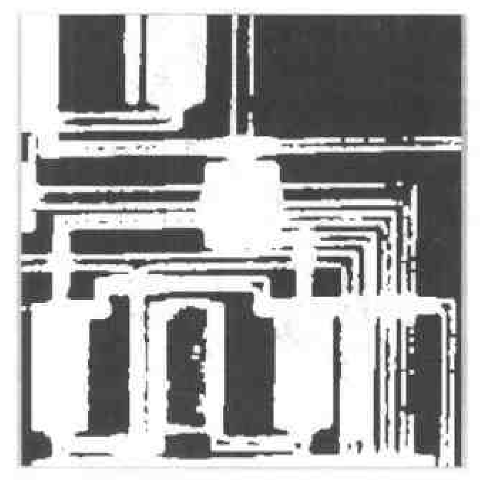


图 10.15 利用 bwmorph 函数开启的结果

10.2 二值形态学进行图像处理的综合应用

10.2.1 噪声滤除

将开启和闭合结合起来可构成形态学噪声滤除器。首先将 saturn.tif 图像二值化，如图 10.16 所示；然后将二值化得到的图像加入椒盐噪声，这种噪声在亮的图像区域内是暗点，而在暗的图像区域中是亮点，如图 10.17 所示；再对有噪声图像进行开启操作，由于这里的结构要素矩阵比噪声的尺寸要大，因而开启的结果是将背景上的噪声点去除了，如图 10.18 所示；最后对前一步得到的图像进行闭合操作，将木星上的噪声去掉了，如图 10.19 所示。实现代码如下：

```
I1=imread('saturn.tif');
I2=im2bw(I1);
figure,imshow(I2)
I3=imnoise(I2, 'salt & pepper');
figure,imshow(I3)
I4= bwmorph(I3, 'open');
figure,imshow(I4)
I5= bwmorph(I4, 'close');
figure,imshow(I5)
```

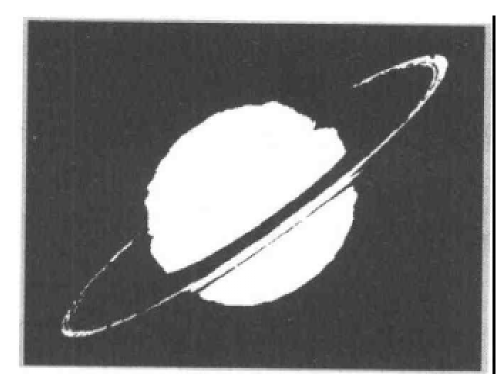


图 10.16　将 saturn.tif 二值化的图像

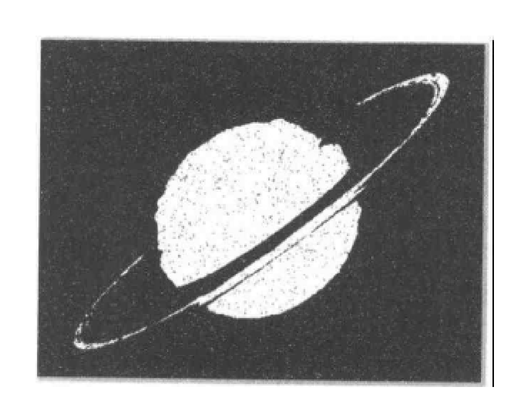


图 10.17　加入椒盐噪声的图像

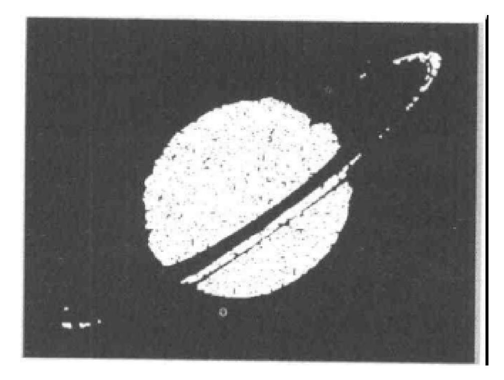


图 10.18　开启操作所得的图像

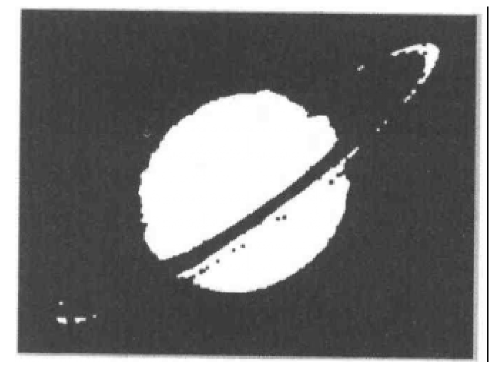


图 10.19　对图 10.18 进行闭合操作所得的图像

虽然采用以上方法达到了去除噪声的目的，但是比较图 10.18 和图 10.16 会发现，处理后的图像在光环上有像素点损失，得到的结果和原图像不完全一样。通过分析可知，出现这样的结果是由于光环上有一些不连续的像素点的尺寸比结构要素矩阵的尺寸小，因此在除噪的过程中也当作噪声一样滤除了。

根据此方法的特点可以知道，此方法适用的图像类型是图像中的对象尺寸都比较大，且没有细小的细节，即对这种类型的图像除噪的效果会比较好。

10.2.2 边界提取

1. 二值图像中的对象

在二值图像中，所谓的对象就是值为 1 的连接在一起的像素的集合。例如，以下所示

的矩阵就代表了包含一个简单的3×3方形对象的二值图像。矩阵中值为 0 的像素表示图像的背景。

0	0	0	0	0	0
0	1	1	1	0	0
0	1	1	1	0	0
0	1	1	1	0	0
0	0	0	0	0	0
0	0	0	0	0	0

2. 4-连接和 8-连接边沿

对于大多数操作来说，二值图像中对象的识别依赖于确定图像中相邻像素是否连接的约定方式。MATLAB 通常使用两种连接方式：4-连接和 8-连接。

在 4-连接边沿约定方式中，所有与用户期望像素点接触的 8 个像素点中，只有垂直方向的 4 个像素均为可能连接的像素，如图 10.20 所示。

在 8-连接边沿约定方式中，所有与用户期望像素点接触的 8 个像素点均为可能连接的像素，如图 10.21 所示。

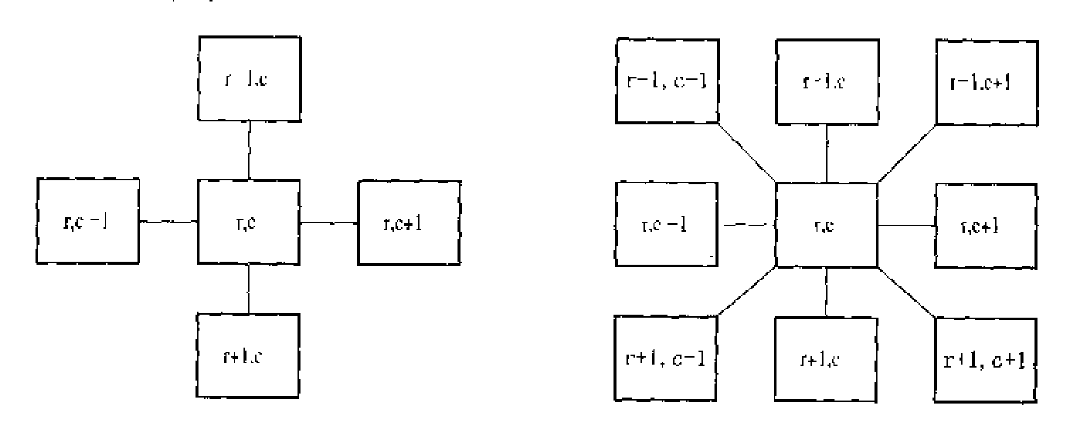


图 10.20 4-连接边沿示例　　　　图 10.21 8-连接边沿示例

根据以上定义，下面的示例矩阵所表示的二值图像在 4-连接边沿约定下包含三个对象，而在 8-连接边沿约定下包含两个对象。

示例矩阵：

0	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0
0	0	0	0	1	1	0	1	1	1
0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	1	1	1

4-连接边沿约定下三个对象分别如下：

0	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	0	0	0	2	2	0	0	0	0
0	0	0	0	2	2	0	3	3	3
0	0	0	0	0	0	0	3	3	3
0	0	0	0	0	0	0	3	3	3

8-连接边沿约定下两个对象分别如下：

0	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0
0	0	0	0	1	1	0	2	2	2
0	0	0	0	0	0	0	2	2	2
0	0	0	0	0	0	0	2	2	2

在 MATLAB 图像处理工具箱中提供了函数 bwperim 来提取二值图像中对象的边界像素。语法如下：

BW2 = bwperim(BW1,n)

确定图像 BW1 中某像素为边界像素的标准是：该像素的值为 1，其邻域中至少有一个像素是非零的。用户可以用 4-连接边沿约定或 8-连接边沿约定来确定对象边界；对应着 n 可取的值有 4 和 8，缺省时采用 4-连接边沿约定。

下面以 blood1.tif 图像的二值化图像(见图 10.22)为例进行边界识别，代码如下：

```
I1=imread(' blood1.tif ');
I2=im2bw(I1);
figure,imshow(I2)
I4= bwperim(I2);
figure,imshow(I4)
```

结果如图 10.23 所示。



图 10.22　blood1.tif 图像的二值化图像



图 10.23　bwperim 函数处理的结果

另外，利用 bwmorph 函数可以得到同样的边界识别效果，代码如下：

```
I1=imread(' blood1.tif ' );
I2=im2bw(I1);
figure,imshow(I2)
I4= bwmorph(I2, ' remove' );
figure,imshow(I4)
```

处理结果如图 10.24 所示。

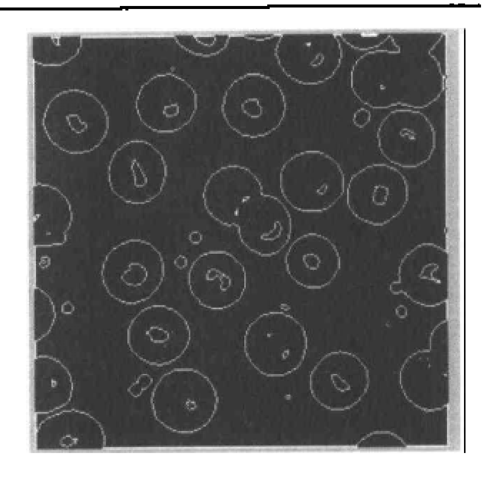
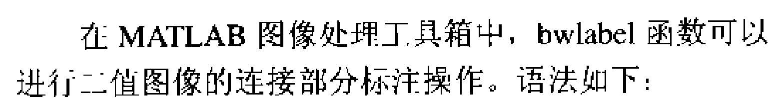


图 10.24　bwmorph 函数处理的结果

10.2.3　对象标注



在 MATLAB 图像处理工具箱中，bwlabel 函数可以进行二值图像的连接部分标注操作。语法如下：

```
L= bwlabel(BW,n)
[L, num]= bwlabel(BW,n)
```

这种形态操作主要用于对二值图像 BW 中各个分离的对象进行标识，BW 可以是 double 型或 uint8 型。用户指定输入图像 BW 和特定的边沿约定类型 n，n 可取的值有 4 和 8，分别对应着 4-连接边沿约定和 8-连接边沿约定，其默认值为 8。调用 bwlabel 函数后，返回一个与输入图像相同大小的数据矩阵 L，L 是 double 型的，而 num 中存储着对象个数的数据。有了这个输出数据矩阵，我们就可以利用它包含的不同的整数值来区分输入图像中的不同对象。

例如，假设有二进制图像如下：

```
I(8,8)=0;
I(2:4,2:3)=1;
I(5:7,4:5)=1;
I(2,6)=1;
I(2:3,7:8)=1
I =  0   0   0   0   0   0   0   0
     0   1   1   0   0   1   1   1
     0   1   1   0   0   0   1   1
     0   1   1   0   0   0   0   0
     0   0   0   1   1   0   0   0
     0   0   0   1   1   0   0   0
     0   0   0   1   1   0   0   0
     0   0   0   0   0   0   0   0
```

调用 bwlabel 函数，指定 4-连接边沿约定

```
[L,num]=bwlabel(I,4)
```

得到的结果如下：

```
L =
        0    0    0    0    0    0    0    0
        0    1    1    0    0    3    3    3
        0    1    1    0    0    0    3    3
        0    1    1    0    0    0    0    0
        0    0    0    2    2    0    0    0
        0    0    0    2    2    0    0    0
        0    0    0    2    2    0    0    0
        0    0    0    0    0    0    0    0
num =3
```

调用 bwlabel 函数，指定 8-连接边沿约定：

```
[L,num]=bwlabel(I,8)或[L,num]=bwlabel(I)
```

得到的结果如下：

```
L =
        0    0    0    0    0    0    0    0
        0    1    1    0    0    2    2    2
        0    1    1    0    0    0    2    2
        0    1    1    0    0    0    0    0
        0    0    0    1    1    0    0    0
        0    0    0    1    1    0    0    0
        0    0    0    1    1    0    0    0
        0    0    0    0    0    0    0    0
num =2
```

我们还可以将 bwlabel 函数与 find 函数结合起来使用，得到所指定对象中各像素的矩阵向量，例如，返回上例中对象 2 中各像素的坐标。

```
[r,c] = find(bwlabel(I)==2)
rc=[r,c]
```

得到的结果如下：

```
rc =
    2    6
    2    7
    3    7
    2    8
    3    8
```

从例中可以看出，输出矩阵不是一个二进制图像。访问该矩阵的一个有效方法是将其显示为伪彩色索引图像，将每个对象都显示成不同的颜色。使得在这样的显示图像中，对象更容易从整个图像中分辨出来。

例如，下面的例子很好地说明了这一点。指定 4-连接边沿约定，代码如下：

```
L=bwlabel(I,4);
```

```
map=[0 0 0;jet(3)];
imshow(L+1,map,'notruesize')
```

显示的图像如图 10.25 所示。或者指定 8-连接边沿约定，代码如下：

```
L=bwlabel(I,8);
map=[0 0 0;jet(3)];
imshow(L+1,map,'notruesize')
```

显示的图像如图 10.26 所示。

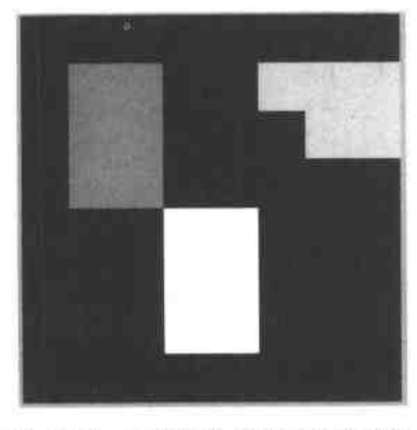


图 10.25 4-连接边沿约定下的对象

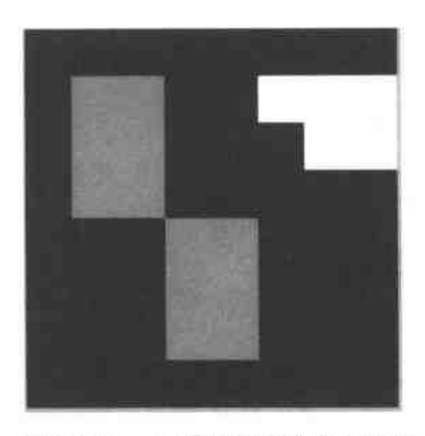


图 10.26 8-连接边沿约定下的对象

10.2.4 图像的特性度量

当我们进行图像处理时，可能会希望获取图像中某些特性的变化信息。例如，在进行放大操作时，可能想知道到底有多少个像素值从 off 状态变成 on 状态，或者想知道图像中的对象数是否改变。为此，MATLAB 的图像处理工作向我们提供了两个函数：bwarea 函数和 bweuler 函数。

1. 图像的面积

bwarea 函数估计二值图像中对象的面积。这里所谓的面积，是指图像前景的大小度量。粗略地讲，面积的大小约等于该图像中值为 on 的像素的数目，但并不是完全等于值为 on 的像素的数目，因为不同方式的像素所占的比重也不同。函数语法很简单，即

```
total = bwarea(BW)
```

其中，BW 可以是 uint8 型或 double 型，total 是 double 型。

bwarea 函数估计二值图像中对象的面积，算法是：把图像中所有值为 on 的像素的面积相加。单个像素的面积是由其 2×2 邻域确定的，有下面 6 种不同的方式，分别表示了 6 种不同的面积。

(1) 有 0 个 on 像素的方式，area=0；
(2) 有 1 个 on 像素的方式，area=1/4；
(3) 有 2 个相邻的 on 像素的方式，area=1/2；
(4) 有 2 个对角的 on 像素的方式，area=3/4；
(5) 有 3 个 on 像素的方式，area=7/8；
(6) 有 4 个 on 像素的方式，area=1。

切记每个像素都是2×2邻域的四个不同位置之一，这意味着，当中间一个像素的值为on，周围的像素都为off时，总的面积为1。

下面以circles.tif图像为例，计算其中对象的面积，代码如下：

```
I= imread('circles.tif');
imshow(I);
area=bwarea(I)
[m,n]=size(I)


area =
        15799
m =
        256
n =
        256
```

还可以通过bwarea函数来计算对图像进行操作前后图像中对象的面积的变化。例如，对circles.tif图像进行腐蚀，代码如下：

```
SE=ones(5)
SE =
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
I2=erode(I,SE);
figure,imshow(I2)
reduce=(bwarea(I)-bwarea(I2))/bwarea(I)
reduce =
            0.1813
```

腐蚀前、后的图像分别如图10.27和图10.28所示。

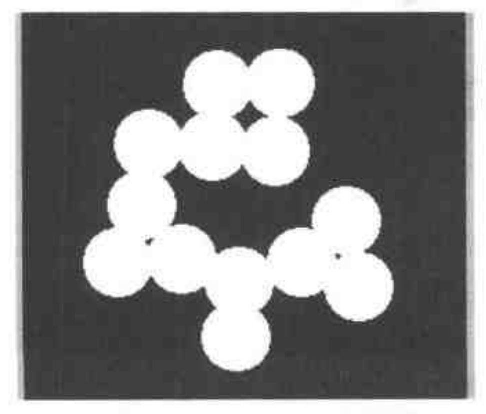


图 10.27　circles.tif 原始图像

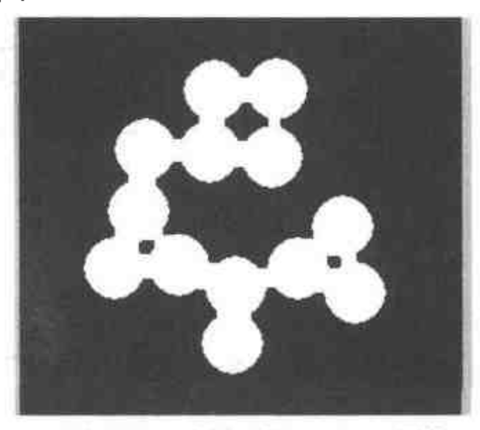


图 10.28　腐蚀后的 circles.tif 图像

另外，imfeature 函数也可以计算对象的面积，但与 bwarea 函数稍有不同，imfeature 函数得到的面积是对象中像素的实际个数。例如，对 circles.tif 图像调用 imfeature 函数，代码如下：

```
area=imfeature(I,'area')
area =
          Area: 15743
```

两个函数计算所得的结果不同，反映了差别。area=imfeature(I,'area')函数调用中，第二个参数指定对图像进行的测度类型，上例中选择了'area'，还可以有其它取值。imfeature 函数的语法如下：

stats = imfeature(L,*measurements*)

stats = imfeature(L,*measurements*,n)

它对标注矩阵 L 中的每个已标注的对象计算一组测度。L 可以通过 bwlabel 函数的计算得到，其中的正整数元素对应着图像中的不同对象，返回值 stats 是个结构数组，显示了由 *measurements* 指定的对每个对象的不同测度。n 可取的值有 4 和 8，分别对应着 4-连接边沿约定和 8-连接边沿约定，缺省值为 8；当 *measurements* 取为'FilledImage'、'FilledArea'和'EulerNumber'时，才需要用到 n。*measurements* 可以是由逗号隔开的字符串列表，可包括以下字符串：'Area'、'Centroid'、'BoundingBox'、'MajorAxisLength'、'MinorAxisLength'、'Eccentricity'、'Orientation'、'Image'、'FilledImage'、'FilledArea'、'ConvexHull'、'ConvexImage'、'ConvexArea'、'EulerNumber'、'Extrema'、'EquivDiameter'、'Solidity'、'Extent'、'PixelList'，也可以是'all'或'basic'。

下面以 text.tif 图像(见图 10.29)为例，运行以下代码：

```
BW = imread('text.tif');
L = bwlabel(BW);
stats = imfeature(L,'all');
stats(23)
```

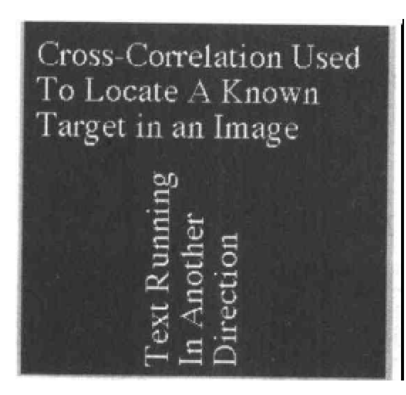


图 10.29 text.tif 图像

运行的结果如下：

```
ans =

                    Area: 89
                Centroid: [95.6742 192.9775]
             BoundingBox: [87.5000 184.5000 16 15]
         MajorAxisLength: 19.9127
```

```
MinorAxisLength: 14.2953
Eccentricity: 0.6961
Orientation: 9.0845
ConvexHull: [28x2 double]
ConvexImage: [15x16 uint8 ]
ConvexArea: 205
Image: [15x16 uint8 ]
FilledImage: [15x16 uint8 ]
FilledArea: 122
EulerNumber: 0
Extrema: [ 8x2 double]
EquivDiameter: 10.6451
Solidity: 0.4341
Extent: 0.3708
PixelList: [89x2 double]
```

2. 欧拉数

在几何理论中，欧拉数是图像的一种拓扑度量。欧拉数等于图像中所有对象的总数减去这些对象中洞孔的数目。函数 bweuler 可以返回二值图像的欧拉数，语法如下：

```
eul = bweuler(BW,n)
```

其中，BW 是输入的二值图像，可以是 double 型或 uint8 型；n 可以取 4 或 8，分别对应着 4-连接边沿约定和 8-连接边沿约定，缺省值为 8；返回值 eul 是 double 型的。

下面以图 10.27 中的 circles.tif 原始图像为例，运行代码如下：

```
I= imread('circles.tif');
imshow(I)
bweuler(I)

ans =

      -2
```

此外，也可调用 imfeature 函数来计算欧拉数，代码如下：

```
number=imfeature(I,' EulerNumber ')
```

可以得到同样的结果：

```
number =
         EulerNumber: -2
```

10.2.5 细化与骨架提取

细化可用两步腐蚀来实现，第一步是正常的腐蚀，但它是有条件的。也就是说，那些被标为可除去的像素点并不立即消去。在第二步中，只将那些消除后并不破坏连通性的点

消除，否则保留。以上每一步都是一个3×3邻域运算。细化将一个曲线形物体细化为一条单像素宽的线，从而图形化地显示出其拓扑性质。

抽骨架是与细化有关的一种运算，也称为中轴变换或焚烧草地技术。中轴是所有与对象在两个或更多非邻接边界点处相切的圆心的轨迹。但抽骨架很少通过在物体内拟合圆来实现。在概念上，中轴可设想成按如下方式形成：想象一片与物体形状相同的草，沿其外围各点同时点火。当火势向内蔓延，向前推进的火线相遇处各点的轨迹就是中轴。

在 MATLAB 中可以调用 bwmorph 函数提取出图像的骨架或细化，实现代码如下：

```
I1= imread('circles.tif');
imshow(I1);
I2 = bwmorph(I1,'skel',Inf);
imshow(I2)
I3= bwmorph(I1,'thin',Inf);
imshow(I3)
```

处理结果如图 10.30 和图 10.31 所示。

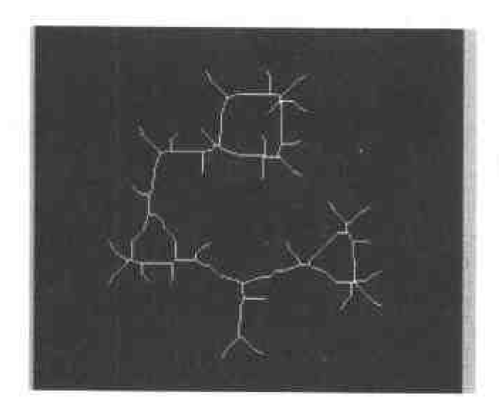


图 10.30 circles.tif 图像的骨架

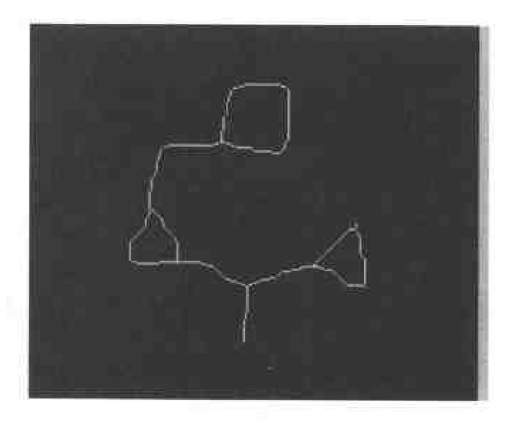


图 10.31 circles.tif 图像细化的结果

比较两幅图可以更加清楚细化和提取骨架之间的差别。

10.2.6 查找表操作

在 MATLAB 中，有些二进制图像操作能够通过查找表非常容易地实现。一个查找表就是一个列矢量，其每个元素均表示边沿中的一种可能的像素组合的返回值。

利用图像处理工具箱中提供的 makelut 函数，我们可以为各种操作创建查找表 lut。makelut 函数可以为 2×2 和 3×3 的边沿创建查找表，分别如图 10.32 和图 10.33 所示。

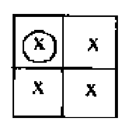


图 10.32 2×2 的边沿

x	x	x
x	ⓧ	x
x	x	x

图 10.33 3×3 的边沿

makelut 函数语法如下：

```
lut = makelut(fun,n)
lut = makelut(fun,n,P1,P2,...)
```

其中，fun 是一个包含了函数名的字符串或 inline 函数对象，此函数以 2×2 或 3×3 的 0-1 矩阵作为输入，返回一个标量；n 的值是 2 或 3，表明 fun 的输入值的尺寸；P1,P2,...也是传递给 fun 的参数；返回值 lut 是一个 double 型的向量。

makelut 函数生成 lut 的方法是：把 2×2 邻域或 3×3 邻域内的所有值都传递给 fun，每次一个，构造含 16 个元素的向量(2×2 邻域)或含 512 个元素的向量(3×3 邻域)。向量中包含了 fun 对每个可能的近邻的输出。

makelut 函数通常与 applylut 函数联合使用，applylut 函数语法如下：

```
A = applylut(BW,lut)
```

其中，BW 是输入的二值图像，可以是 uint8 型或 double 型；lut 是由 makelut 返回的 16 元素的向量，或 512 元素的向量，可以是 uint8 型或 double 型；A 的值取决于 lut 中的值，如果 lut 中的值全部是 1 和 0，则 A 是二值图像。如果 lut 中的所有元素都是 0 到 255 之间的整数，则不管 lut 是什么类型的，A 都是 uint8 型的；否则，A 是 double 型的。

applylut 对二值图像执行邻域操作，生成对 lut 的索引矩阵，然后用 lut 中的实际值替换索引值。具体的算法取决于所采用的邻域是 2×2 的，还是 3×3 的。

1. 2×2 邻域

在这种情形下，lut 的长度为 16，每个邻域中都有 4 个像素，每个像素都有 2 个可能的状态，所以交换的总次数为：$2^4=16$。Applylut 将二值图像与矩阵

8	2
4	1

进行卷积，生成索引矩阵。卷积结果中的值都是[0,15]范围内的整数值，applylut 使用卷积中与输入二值图像矩阵相同大小的中心部分，将每个值都加 1，使得范围变为[1,16]。然后，将索引矩阵中每个位置上的值用 lut 中索引值所指向的值替换，构造出 A。

2. 3×3 邻域

在这种情形下，lut 的长度为 512，每个邻域中都有 9 个像素，每个像素都有 2 个可能的状态，所以交换的总次数为：$2^9=512$。Applylut 将二值图像与矩阵

256	32	4
128	16	2
64	8	1

进行卷积，生成索引矩阵。卷积结果中的值都是[0, 511]范围内的整数值，applylut 使用卷积中与输入二值图像矩阵相同大小的中心部分，将每个值都加 1，使得范围变为[1,512]。然后，将索引矩阵中每个位置上的值用 lut 中索引值所指向的值替换，构造出 A。

下面是一个通过查找表来修改包含文本的图像 text.tif 的例子，如图 3.34 所示。首先利用嵌入函数创建查找表，代码如下：

```
f=inline('sum(x(:)) == 4');
lut = makelut(f,2)
```

```
lut =
        0
        0
        0
        0
        0
        0
        0
        0
        0
        0
        0
        0
        0
        0
        0
        1
```

然后，调用 applylut 函数利用查找表 lut 对图像进行操作，代码如下：

```
I1 = imread('text.tif');
I2 = applylut(I1,lut);
imshow(I1)
figure, imshow(I2)
```

最后的结果如图 10.35 所示。本例使用了 2×2 邻域来执行腐蚀。当某输入像素的四个相邻像素的值都为 on 时，该像素的输出值才为 on。

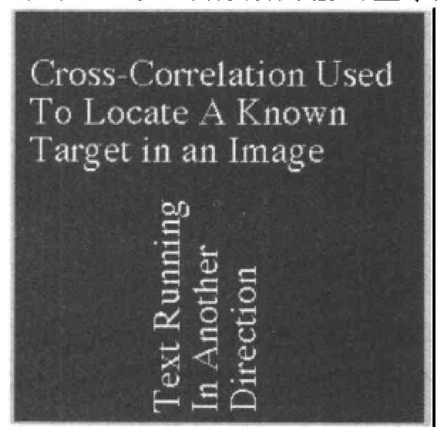
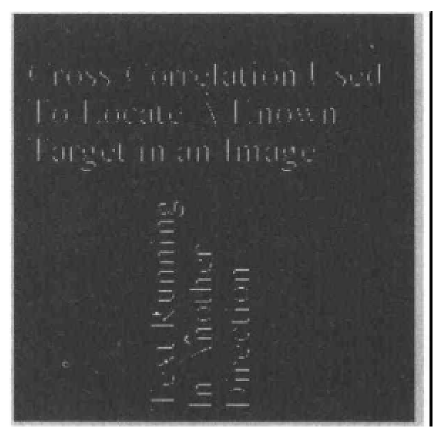


图 10.34 text.tif 图像


图 10.35 应用查找表的结果

10.2.7 区域填充

此二值图像操作与其它操作的不同之处在于它对背景像素进行操作，而不对前景像素

进行操作。在 MATLAB 中由 bwfill 函数来实现，语法如下：

```
BW2 = bwfill(BW1,c,r,n)
BW2 = bwfill(BW1,n)
[BW2,idx] = bwfill(...)
BW2 = bwfill(x,y,BW1,xi,yi,n)
[x,y,BW2,idx,xi,yi] = bwfill(...)
BW2 = bwfill(BW1,'holes',n)
[BW2,idx] = bwfill(BW1,'holes',n)
```

其中：

BW2 = bwfill(BW1,c,r,n)表示对图像 BW1 进行区域填充，(r,c)为填充的起始坐标，如果是等长的向量，则从(r(k),c(k))开始进行平行填充。n 为区域的连通数，可取值为 4 或 8，分别对应着前景的 4 邻域边沿约定和 8-邻域边沿约定，缺省值为 8。BW1 的前景由所有值为 1 的像素组成。

BW2 = bwfill(BW1,n)表示将图像 BW1 显示在屏幕上，由用户用鼠标交互地选取填充的起始点。如果忽略了 BW1，则函数对当前坐标轴内的图像进行操作。点击常规按钮来增加点，按下 Backspace 键或 Delete 键删除前面选取的点，按下 Shift 键并点击、右击或双击可以确定终点，并开始填充；按下 Return 键，则终止选点。

[BW2,idx] = bwfill(...)表示返回填充点的线性索引

BW2 = bwfill(x,y,BW1,xi,yi,n)表示用向量 x，y 为 BW1 设置非缺省的空间坐标系，xi 和 yi 是标量或等长的向量，指定起始填充点在坐标系中的位置。

[x,y,BW2,idx,xi,yi] = bwfill(...)表示返回的输出图像存放在 BW2 中，横、纵数据分别返回到 x、y 中，所有填充像素点的线性索引返回到 idx 中，填充起始点返回到 xi、yi 中。

BW2 = bwfill(BW1,'holes',n)表示自动确定哪些像素点在对象孔洞中，然后将这些像素点的值由 0 改为 1。n 的缺省值为 8。

[BW2,idx] = bwfill(BW1,'holes',n)表示返回函数填充的所有像素点的线性索引。

如果函数没有输出值，则将填充的结果显示出来，特别要说明的是当前景是 4-邻域边沿约定时，背景是 8-邻域边沿约定；反之亦然。输入图像可以是 double 型的或 uint8 型的，输出图像是 uint8 型的。

例10.2.1 用形态学的方法填充图像中的孔洞。

首先，对blood1.tif图像求反，使前景像素值变成1，代码如下：

```
I = imread('blood1.tif');
BW1 = ~im2bw(I);
```

然后，进行填充，代码如下：

```
BW2 = bwfill(BW1,'holes');
subplot(1,2,1),imshow(BW1)
subplot(1,2,2),imshow(BW2)
```

结果如图 10.36 和图 10.37 所示。

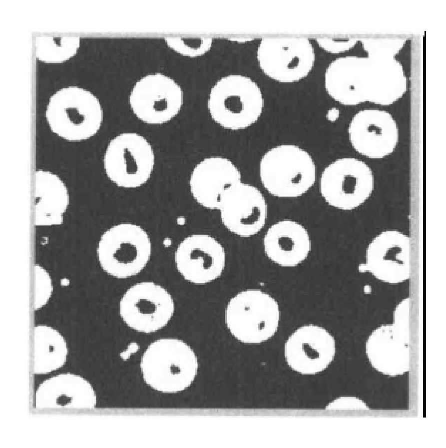



图 10.36 blood1.tif 求反的图像

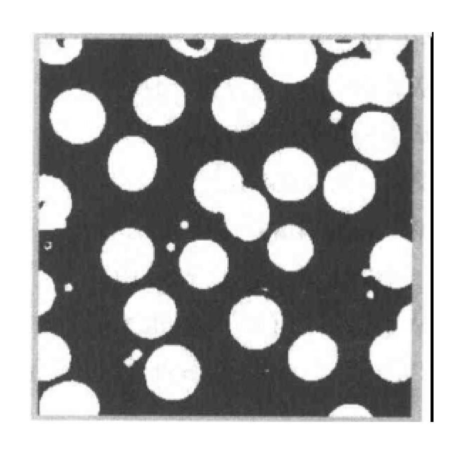


图 10.37 填充图像中的孔洞

10.2.8 对象提取

对象提取在模式识别等很多应用领域中都很有用，因此也是图像处理中的重要内容。在 MATLAB 中可以由 bwselect 函数来实现，语法如下：

```
BW2 = bwselect(BW1,c,r,n)
BW2 = bwselect(BW1,n)
[BW2,idx] = bwselect(...)
BW2 = bwselect(x,y,BW1,xi,yi,n)
[x,y,BW2,idx,xi,yi] = bwselect(...)
```

其中：

BW2 = bwselect(BW1,c,r,n)表示返回二值图像，其中包含了与像素(r,c)交叠的对象。r、c 可以是标量或等长的向量，如果 r、c 是向量，则 BW2 包含了与各像素(r(k),c(k))交叠的对象集。n 可以取 4 或 8，分别对应着对象的 4-邻域边沿约定和 8-邻域边沿约定，缺省值为 8。对象由所有值为 1 的像素组成。

BW2 = bwselect(BW1,n)表示将图像 BW1 显示在屏幕上，由用户用鼠标交互地选取(r,c)坐标，如果忽略了 BW1，则函数对当前坐标轴内的图像进行操作 点击常规按钮来增加点，按下 Backspace 键或 Delete 键删除前面选取的点，按下 Shift 键并点击、右击或双击可以确定终点，并开始填充；按下 Return 键，则终止选点。

[BW2,idx] = bwselect(...)表示返回所选对象中像素点的线性索引。

BW2 = bwselect(x,y,BW1,xi,yi,n)表示用向量 x、y 为 BW1 设置非缺省的空间坐标系，xi 和 yi 是标量或等长的向量，指定了对象在坐标系中的位置。

[x,y,BW2,idx,xi,yi] = bwselect(...)表示返回的输出图像存放在 BW2 中，横、纵数据分别返回到 x、y 中，所选对象中所有像素点的线性索引返回到 idx 中，所指定的空间坐标返回到 xi、yi 中。

如果函数没有输出值，则将填充的结果显示出来。输入图像可以是 double 型的或 uint8 型的，输出图像是 uint8 型的。

例 10.2.2 以 text.tif 图像(见图 10.38)为例，提取文本图像中的字符对象，代码如下：

```
BW1 = imread('text.tif');
c = [16 90 144];
r = [85 197 247];
BW2 = bwselect(BW1,c,r,4);
imshow(BW1)
figure, imshow(BW2)
```

结果如图 10.39 所示。

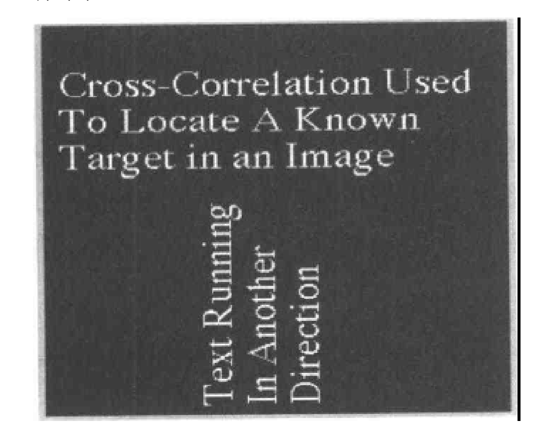


图 10.38　text.tif 图像

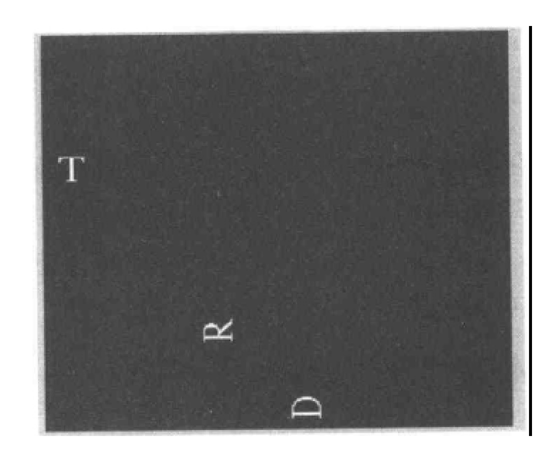


图 10.39　提取字符对象

综合应用篇

第十一章 综合应用实例

在实际应用中，所需要解决的问题通常不可能仅仅通过某一种图像处理方法就能得到比较理想的效果，而是需要综合应用多种方法，通过适当的组合才能达到图像处理的目的。因此，本章将针对几个典型实例，结合使用前几章中介绍的图像处理方法，对处理的过程进行详细描述。

本章主要内容：

- ★ 对不均匀亮度的校正
- ★ 基于特征的逻辑
- ★ 对钢纹(steel grain)的区域标识

11.1 对不均匀亮度的校正

由于拍摄过程的原因造成图像中亮度不均匀的现象是很常见的。在这种情况下，再对图像进行边缘检测、模式识别等，其它工作就会受到影响，因此要首先对不均匀亮度进行校正。

本例将详细讲解对一幅大米粒图像的不均匀亮度的校正。首先绘制原图：

```
I1=imread('rice.tif');
figure,imshow(I1)
```

上述语句绘制出的原始图像如图 11.1，可以看出图像上中部的背景比下部的背景要亮得多。

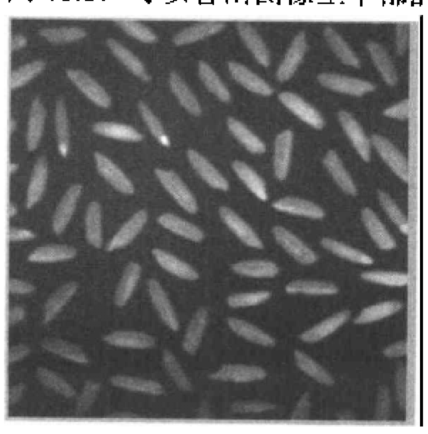


图 11.1 rice.tif 原始图像

第一步：使用 blkproc 命令对背景的亮度进行粗略的估计，确定图像中每个 32×32 方块的最小值。下面画出了这个粗略估计的表面图，在 MATLAB 图形窗口中可以拖动坐标轴，使图形旋转。实现代码如下：

```
I2=im2double(I1);
bg32=blkproc(I2,[32 32],'min(x(:))');
surf(bg32),rotate3d on
```

图形窗口如图 11.2 所示。

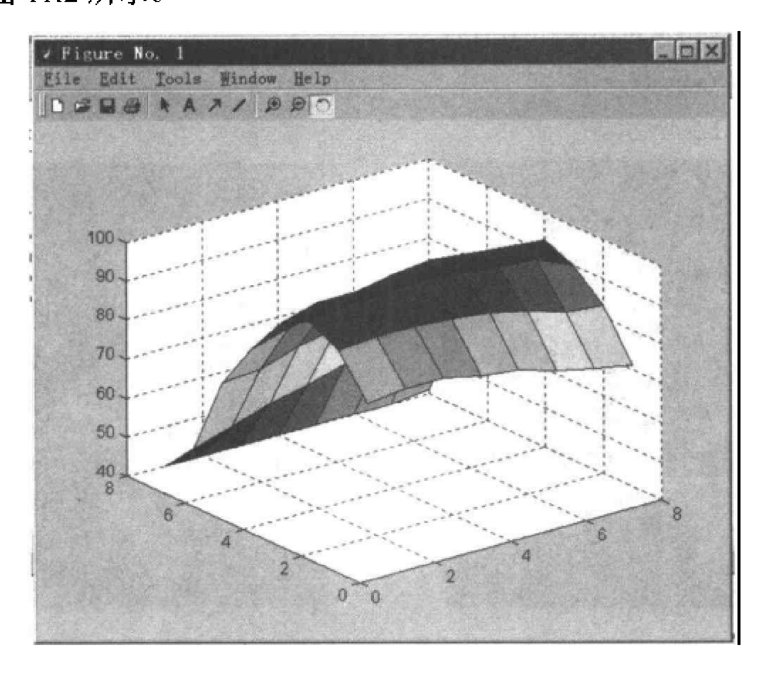


图 11.2　对背景亮度的粗略估计的表面图

第二步：使用 imresize 函数将粗略估计的表面图扩展到与原始图像一样大小。使用三次插值对数据进行平滑处理。代码如下：

```
bg256=imresize(bg32,[256 256],'bicubic');
figure,imshow(bg256)
```

处理效果如图 11.3 所示。

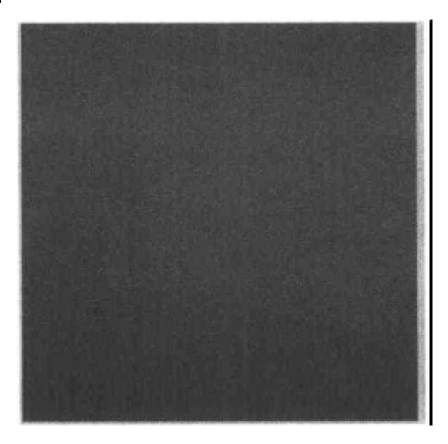


图 11.3　imresize 函数处理结果

第三步：从原始图像中减去前面计算出来的亮度，校正不均匀性，但此步操作同时会使大米粒也变暗。代码如下：

```
d=I2-bg256;
figure,imshow(d)
```

处理效果如图 11.4 所示。

第四步：将像素的灰度值调整到整个灰度级，使得大米粒变得较亮，具有更丰富的细节。代码如下：

```
adjust=imadjust(d,[0 max(d(:))],[0 1],1);
figure,imshow(adjust)
```

处理效果如图 11.5 所示。

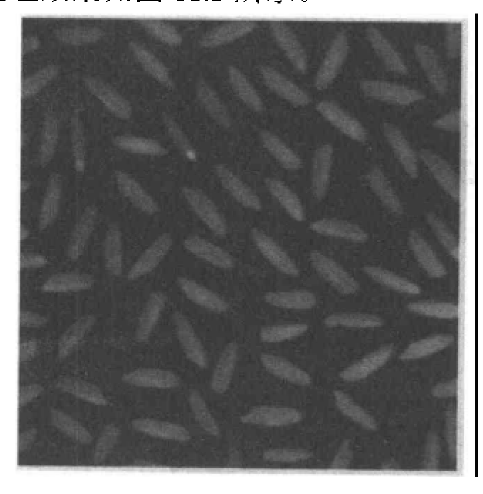


图 11.4 从原始图像减去亮度的结果

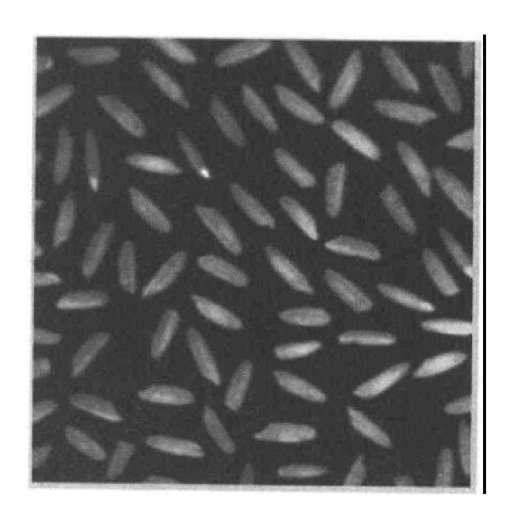


图 11.5 灰度调整后的最终处理结果

将图 11.5 与原始图像(见图 11.1)进行比较，可以明显看出，处理后的结果中背景亮度分布均匀，米粒与背景之间的对比明显，米粒清晰，易于辨识处理。

11.2 基于特征的逻辑

下面分别以合成二值图像和细菌的显微图像为例，实现基于特征的逻辑。

(1) 本例详细讲解了将基于特征的逻辑应用于合成二值图像。在图 11.6 和图 11.7 中，找出 dots 图像的特征和 box 图像的特征相交叠的部分。代码如下：

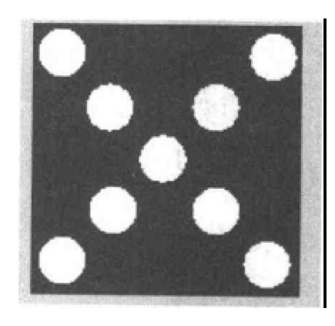


图 11.6 dots 图

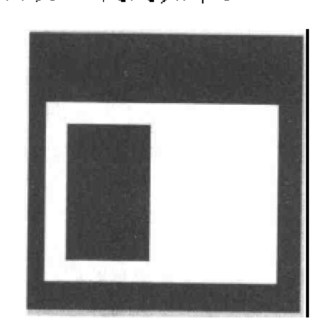


图 11.7 box 图

```
load imdemos dots box
imshow(dots);
figure,imshow(box)
```

第一步：对两幅图像进行逻辑与操作，辨识在两幅图像中值均为 on 的像素，代码如下：

```
logical_and=box&dots;
figure,imshow(logical_and);
```

处理后的图像如图 11.8 所示。

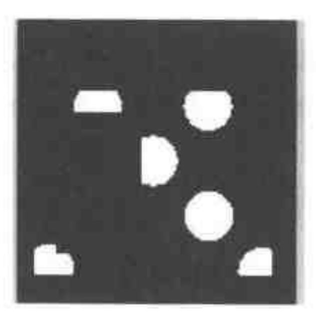




图 11.8 两幅图逻辑与的结果

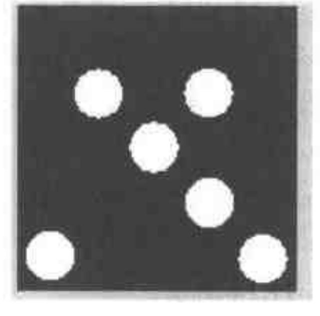


图 11.9 基于特征的逻辑与的结果

第二步：将前一步逻辑与中值为 on 的像素作为 bwselect 函数的输入，实现基于特征的逻辑与，代码如下：

```
[r,c]=find(logical_and);
feature_and=bwselect(dots,c,r);
figure,imshow(feature_and)
```

处理后的图像如图 11.9 所示。

(2) 本例讲解了将基于特征的逻辑应用于细菌的显微图像。在图 11.10 中，我们要辨识出哪些细菌包含一个或多个亮颗粒，确定包含颗粒的细菌的个数，而不是确定颗粒的个数，因为有一些细菌可能包含多个颗粒。代码如下：

```
load imdemos bacteria
figure,imshow(bacteria)
```

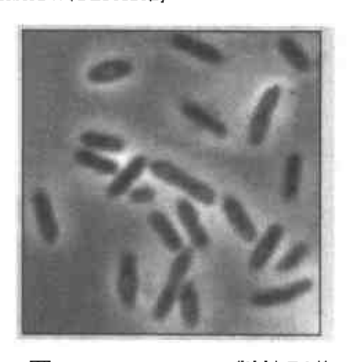


图 11.10 bacteria 原始图像

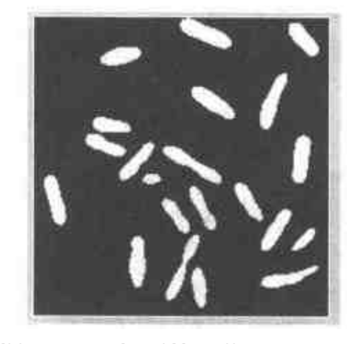


图 11.11 得到的二值分割图像

第一步：对图像取阈值 100，将细菌进行分割，得到二值分割图像，代码如下：

```
bact_bw=~(bacteria>100);
figure,imshow(bact_bw)
```

处理结果如图 11.11 所示。

第二步：对原始图像应用拉普拉斯滤波器，对滤波结果取阈值-4，然后与上一步中得到的二值分割图像进行逻辑结合，得到的图像中显示了细菌细胞中的颗粒。代码如下：

```
filtered=filter2(fspecial('laplacian'),bacteria);
filtered_bw=(filtered>-4);
bact_granules=filtered_bw&bact_bw;
figure,imshow(filtered,[])
figure,imshow(bact_granules)
```

处理结果如图 11.12 和图 11.13 所示。

第三步：腐蚀第一步中得到的分割图像，在结果中选择 bact_granules 为 0 的区域，从而将颗粒分离开来，腐蚀后可以避免检测到细菌细胞边缘上的点。代码如下：

```
granules=erode(bact_bw);
granules=granules&(bact_granules==0);
figure,imshow(granules)
```

处理结果如图 11.14 所示。

第四步：辨识取过阈值的简单图像中有颗粒的对象。代码如下：

```
[r,c]=find(granules);
result=bwselect(bact_bw,c,r);
figure,imshow(bacteria)
figure,imshow(result)
```

处理结果如图 11.15 所示。

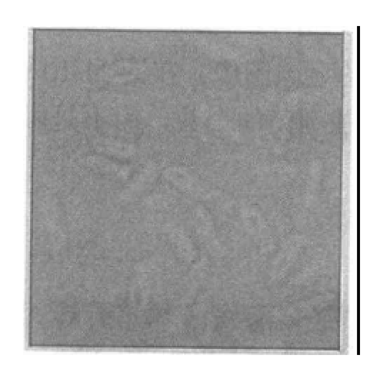


图 11.12　应用拉普拉斯滤波器的结果

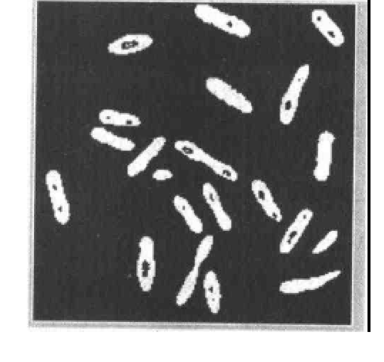


图 11.13　逻辑结合的结果

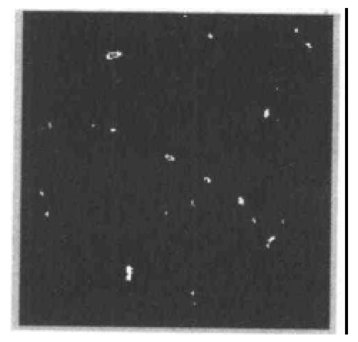


图 11.14　检测到的亮颗粒

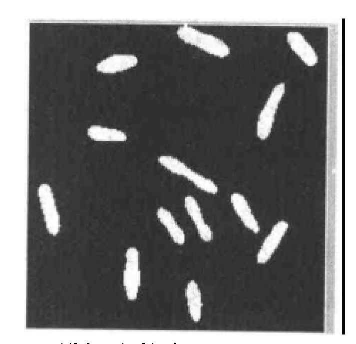


图 11.15　辨识出的有颗粒的细菌细胞

11.3　对钢纹(steel grain)的区域标识

本例中通过取两次阈值，骨架化及二值逻辑非操作，确定了钢的显微图像中颗粒的边界，标识了不同的颗粒(steel 的原始图像见图 11.16)。

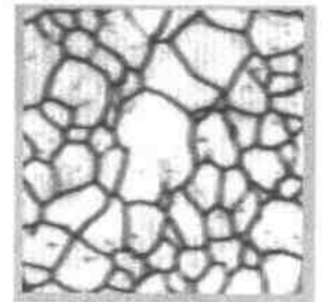


图 11.16　steel 的原始图像

第一步：对钢的图像取两个不同的阈值 70 和 210，代码如下：

```
load imdemos steel;
figure,imshow(steel)
bw_70=steel>70;
bw_210=steel>210;
figure,imshow(bw_70)
figure,imshow(bw_210)
```

处理结果如图 11.17 和图 11.18 所示。



图 11.17　取阈值 70 的图像



图 11.18　取阈值 210 的图像

第二步：用图 11.17 中的黑色点来选取图 11.18 的求反图像中的白色对象，这样可以有效地去除图 11.18 中的较小区域，代码如下：

```
[r,c]=find(bw_70==0);
bw_clean=bwselect(~bw_210,c,r,8);
figure,imshow(bw_clean)
```

处理结果如图 11.19 所示。



图 11.19　去除了图 11.18 中的较小区域的结果

第三步：将去除了较小区域的结果骨架化，然后从骨架化的图像中剪去尖刺像素，代码如下：

```
bw_skel=bwmorph(bw_clean,'skel',6);
figure,imshow(bw_skel)
bw_pruned=bwmorph(bw_skel,'spur',8);
figure,imshow(bw_pruned)
```

处理结果如图 11.20 和图 11.21 所示。

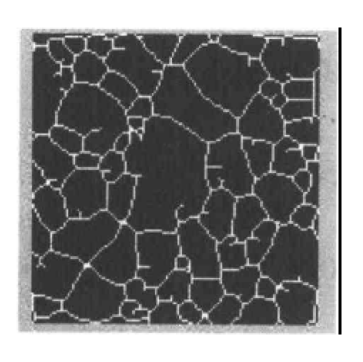


图 11.20 骨架化的结果

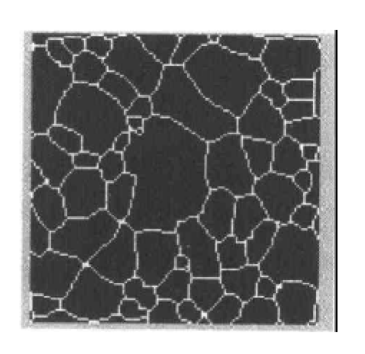


图 11.21 剪刺后的结果

第四步：颗粒的边界图就是剪刺图像的逻辑非，代码如下：

```
grain_boundaries=~bw_pruned;
figure,imshow(grain_boundaries)
```

处理结果如图 11.22 所示。

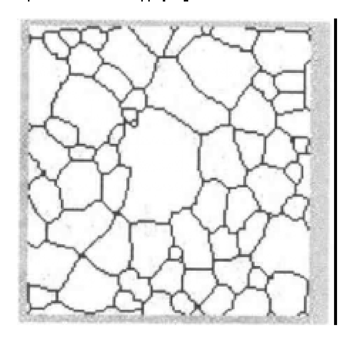


图 11.22 剪刺图像的逻辑非

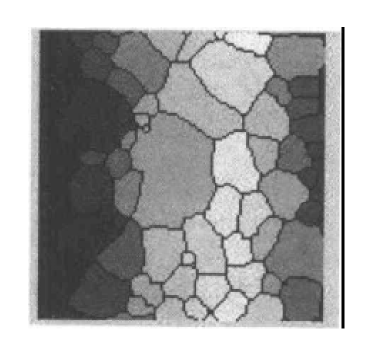


图 11.23 伪彩色图像

第五步：将对象映射成不同的颜色显示出来。Bwlabel 命令可以将二值结果中的不同对象标识出来，代码如下：

```
[labeled,N]=bwlabel(grain_boundaries,4);
colored=ind2rgb(labeled+1,[0 0 0;jet(N)]);
figure,imshow(colored)
```

处理结果如图 11.23 所示。

综合应用篇

第十二章　图形用户界面设计

图形用户界面(GUI)在 MATLAB 程序开发中起着举足轻重的作用。一个好的界面不仅有利于用户快速掌握程序的操作流程，有效地使用程序，也有利于开发者展示 MATLAB 平台下的开发技术。GUI 在科研实践和工程实践中有着广泛的应用，尤其是在图形处理技术、人工智能技术等方面。用户也许对此深有体会：在 MATLAB 的 DEMO 中，每一个实例都是 GUI 成功应用的例子。一个个栩栩如生的 DEMO 实例逐步引导用户了解、熟悉 MATLAB 的使用。读者是否也萌发了为自己的程序也创建一个 GUI 的念头。那么，在 MATLAB 下怎么创建 GUI？什么样的 GUI 才是比较成功的？希望本章能够给读者一个满意的答复。

在科研实践中，越来越多的人放弃 VC，选用 MATLAB 来开发界面。这种现象的出现缘于以下两方面的原因：① 开发者的重点是放在核心算法的设计，GUI 只是为了向客户演示技术的方便，相比较而言，VC 太复杂，需要开发者做大量的底层工作；② 随着 MATLAB 在工程应用中的推广，许多核心的算法是用 MATLAB 实现的，因此，选用 MATLAB 的界面可以做到代码的无缝连接，不需涉及到复杂的接口技术。

本章的内容大致是这样安排的。首先，介绍一个简单实例。通过该实例，读者会对 MATLAB 下的用户界面开发过程有一个大致的了解，同时初步掌握 MATLAB 图形用户界面开发环境 GUIDE 的使用。一个成功的 GUI 必须建立在精巧的设计之上。因此，本章接下来侧重分析 GUI 的设计过程中需注意的一些问题，也可以说是开发者必须遵循的一些设计原则。只要遵循这些原则，相信你所开发出的界面会很容易被用户所接受。本章最后用了大量篇幅结合实例对 GUI 实现中的技术细节作了介绍。开发者如果掌握了这些细节，就可以在用户界面上自由发挥，使图形界面具有更多的个性。

在开始本章之前，读者需要对图形句柄的概念有一个大致了解。如果有用 VC、VB 开发界面经验的话，当然更好。因为 MATLAB 下的界面开发也使用了当今的主流图形开发技术，开发环境大同小异，很多地方有相似之处。在使用中往往可以触类旁通。如果你是新入门的读者，也无须顾虑。相对其它章节而言，本章独立成篇，深入浅出的介绍会让你在读完本章之后，晋级到高手的行列。

本章主要内容：

- ★ GUIDE 开发环境介绍
- ★ GUI 设计
- ★ GUI 实现

12.1　GUIDE 开发环境介绍

既然能够用 MATLAB 求解数学问题，当然也可以用 MATLAB 把图形用户界面画出来。解决这个问题的方法便是使用 MATLAB 中的 GUIDE。

GUIDE 实际上是一套 MATLAB 工具集，它由五个部分组成：属性编辑器、控件面板、回调函数编辑器、调整工具和菜单编辑器。有了这些工具，我们可以在短短几分钟内设计出一个漂亮的 GUI。当然，这只是问题的第一步。一旦有了界面，后面的工作便是通过 GUIDE 的回调函数编辑器编写响应函数代码，来响应用户的界面操作。下面的例子便是将一条函数曲线显示在 GUI 中，然后用 GUIDE 向这个 GUI 中添加按钮，编写按钮的回调函数。于是，通过按钮可以控制函数曲线的显示过程。这当然只是一个简单的用户交互界面，但是通过它，你可以明白什么是 GUIDE，如何用 GUIDE。

例 12.1.1　绘制函数曲线：t*sin(t)。

在 MATLAB 命令行中输入以下命令：

```
t = 0:0.1:20;
plot(t,t.*sin(t))
%绘制函数曲线，生成一个图形 Figure
get(gcf, 'Color')
%gcf 函数返回当前 Figure 的句柄，get 得到当前 Figure 的颜色属性
ans =0.8000 0.8000 0.8000
%MATLAB 中的所有颜色以 RGB 的形式存储，[1,1,1]是白色，[0,0,0]是黑色。此时，可以用命
%令行的形式激活属性编辑器
propedit(gcf)
```

命令行产生的图形与激活的属性编辑器如图 12.1 所示。

在属性编辑器的菜单【选项(options)】中，选中【show object browser】，从对象浏览框中，可以发现 Figure 所包含的所有对象。这些对象以树形结构排列，将这个对象树全部展开后，你将会发现图形 Figure 中出现的任何一个可视部件都能够在这个对象树中找到它所对应的对象。选中一个对象，下面的属性列中将会出现该对象所具有的全部属性，以及属性允许的取值。比如图形 Figure 的“color”属性，允许的取值是颜色表中的任何一种颜色。

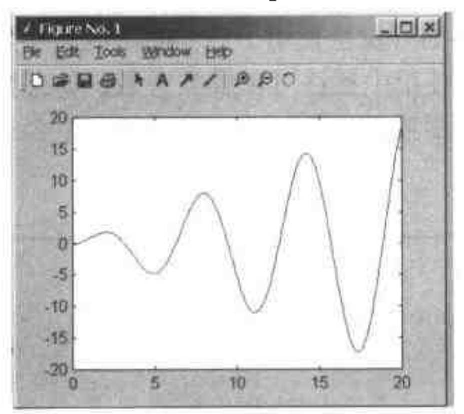




图 12.1　命令行产生的图形(左图)与激活的属性编辑器(右图)

12.1.1　控制面板添加按钮

在属性编辑器的【工具(tools)】中选择【Control Panel】，将会弹出控制面板的界面(见图 12.2)。

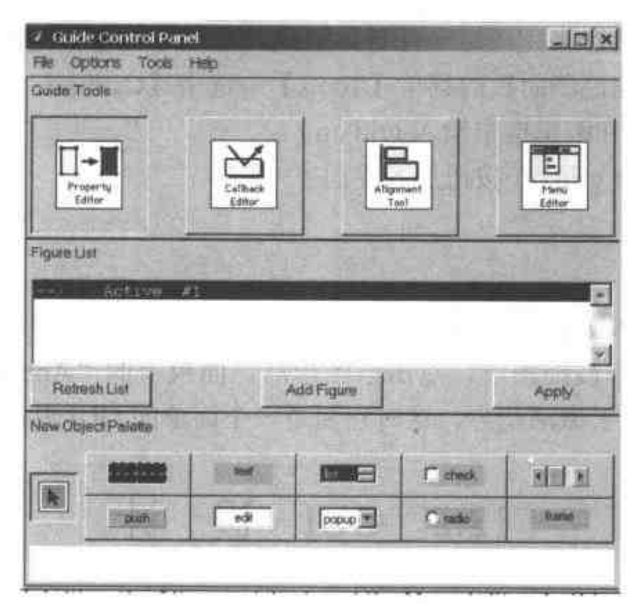


图 12.2　控制面板

该控制面板分为三个部分：

(1) 上面部分是 GUIDE 工具列表；

(2) 中间部分是 GUIDE 所控制的图形(Figure)列表；

(3) 最下面一部分是可以向 Figure 中添加的对象，包括按钮、文本框、复选框、列表框等等。

从该控制面板的界面，可以发现【Property Editor】处于“Push in”状态。这是因为属性编辑器已被打开，并被摆在桌面上。

中间的图形列表可以显示各个图形(当前只有一个图形)的状态。当该图形处于“Controlled”状态时，用户可以随意地移动被控图形上的对象，比如按钮、坐标轴等。GUIDE 会自动计算这些对象移动后所处的位置，并把位置坐标的数值放入 MATLAB 的代码中。当设计者将图形上的一切都移动到位后，可以把图形的状态切换至“Activated”。这时，对图形中对象的拖拉无效。

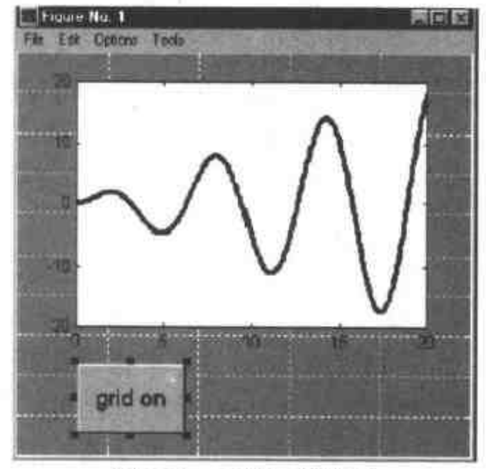


图 12.3　函数曲线界面

从控制面板下部点击待添加的对象，这里是“button”。当鼠标移动到被控图形区域时，鼠标箭头将变成一个带加号的箭头。此时，便可通过拖拉鼠标，改变按钮的大小，直到符合你的需要。按照以上步骤，便可顺利地将一个按钮添加到图形界面上。这时，该按钮仍处于被选中状态。接下来，还需要在属性编辑器中修改按钮的“string”属性为“grid on”(注意，不要漏掉双引号)。于是，出现图 12.3

所示界面。

12.1.2 使用回调函数编辑器编写回调函数

现在，我们只是把一个按钮放到了界面上，还没有定义按下该按钮后应执行的动作。下面使用回调函数编辑器来完成这一任务。首先，在按钮对象被选中的情况下，激活回调函数编辑器。方法是在控制面板的菜单【Tools】中选中【Callback Editor】。然后在下拉式菜单中选中 callback，在编辑框中输入 grid on。

重复以上步骤，将 grid off 按钮加入界面。

12.1.3 激活图形

将图形从编辑状态切换到激活状态，使前几步的操作生效。具体方法是回到控制面板下，在图形列表中选中被控图形，点击该图形后，面板上的“Apply”按钮有效，再点击 Apply 按钮，图形便处于激活状态。这时它便是一个简单的 GUI 界面。

12.2 GUI 设 计

设计包括编写源代码之前的所有工作。设计者必须重视设计工作，不要急于编写源代码。否则，编程员思路不清晰，开发过程中会走很多弯路，因而开发效率低下，并且设计出的 GUI 也不一定让人满意。那么如何进行成功的设计，如何避免上述可能出现的问题呢？本节将从两个角度予以探讨：一方面论述设计 GUI 的指导原则，另一方面介绍设计过程中应遵循的步骤。以上两点已被许多 GUI 开发者的实践所证明。

12.2.1 指导原则

可以用几个字来概括优秀设计的标准：简洁性，一致性，熟悉性。

简洁性意味着界面简单明了，直接清晰。用户可以很快提取出对自己有用的信息。

一致性意味着设计系统的各个环节应遵从统一的、简单的规则，保证不出现例外情况。

熟悉性意味着系统设计时尽量保持与该类流行界面的相似，比如 VC 的对话框风格、视图的风格、MATLAB 的 Figure 风格，这样用户操作起来就容易上手，误操作率比较低。

以上三点也许过于抽象，具体过程中不易操作。其实，只要在设计整个过程中贯穿以人为本的设计理念，设计出的界面自然会达到以上要求。设计者应该经常考虑以下两个问题：

(1) 用户在没有熟悉界面之前，完成一个操作需要多长时间？

(2) 用户在熟悉界面后，完成一个普通操作需要多长时间？

简洁性、一致性、熟悉性围绕着上述两个问题展开。它们有时也存在着矛盾的地方，比如会为了一致性而牺牲简洁性。这种取舍的标准应着眼于用户，要看是否便于用户操作。用户应该是设计者心中的上帝。

1. 简洁性

简洁是一种直观的美，它是设计者所要达到的主要目标，通过 GUIDE，我们很容易向界面上添加许多功能。但功能强大并不一定代表成功，有时增加一些功能反而显得画蛇添足。GUI 本来是从图像的角度帮助用户理解客观世界的，但是缤纷芜杂的表面现象却容易掩盖事物的真相，影响用户做出正确的判断，这当然是不可取的。设计者一定要警惕这种盲目追求功能强大的心理。下面举出的一些例子，也许会给设计者带来一些启发。

1) 注重形式

例 12.2.1 三维形体的观察。

该例子对比显示了一个三维形体加网格和不加网格的视觉效果(见图 12.4 和图 12.5)。通过对比，我们可以看到网格画蛇添足，遮挡了部分视线。

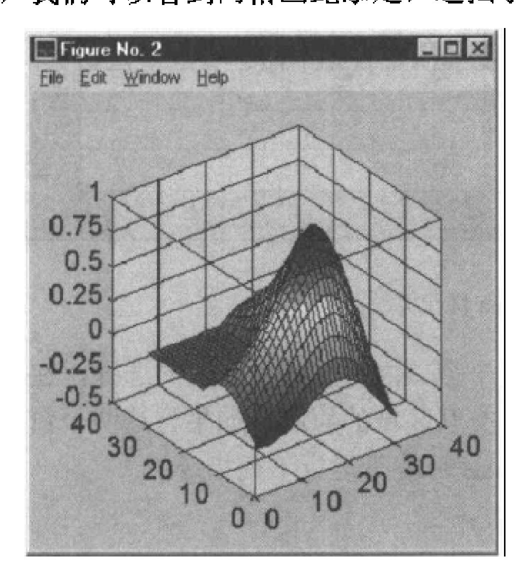


图 12.4 添加网格的视觉效果　　图 12.5 取消网格的视觉效果

图像往往给人一种直观的感受，告诉人们事物发展的趋势，数字有助于将这种趋势量化，增强人们对发展趋势的认识。但是这并不意味着图像必须借助于数字。比较上面两个界面，你会发现图 12.4 的网格并没有起到它应起的作用，相反还是个累赘。为什么不把它去掉？相比之下，图 12.5 就好看多了，一眼望去，马上可以对这个三维图形的凸凹形成深刻的印象，剩下的问题就是如何求解这个图形对应的函数中含有的全局极大点了。

2) 缩小交互区域

如果能在一个界面中完成的事情，就坚决不要放在两个界面中去完成。这也是操作简洁性的一种间接体现。

例 12.2.2 banana 优化算法。

在 MATLAB 运行环境下，运行下列指令：

?bandemo

该演示程序是为了显示各种最优化算法对 banana 函数的处理效果。banana 函数是最优化理论中的一个著名问题。由于该曲线中间的曲率半径特别小，因而很多最优化算法不会收敛到极值点。因此用这个函数来检验算法的收敛性特别有效。banana 优化算法如图 12.6 所示。

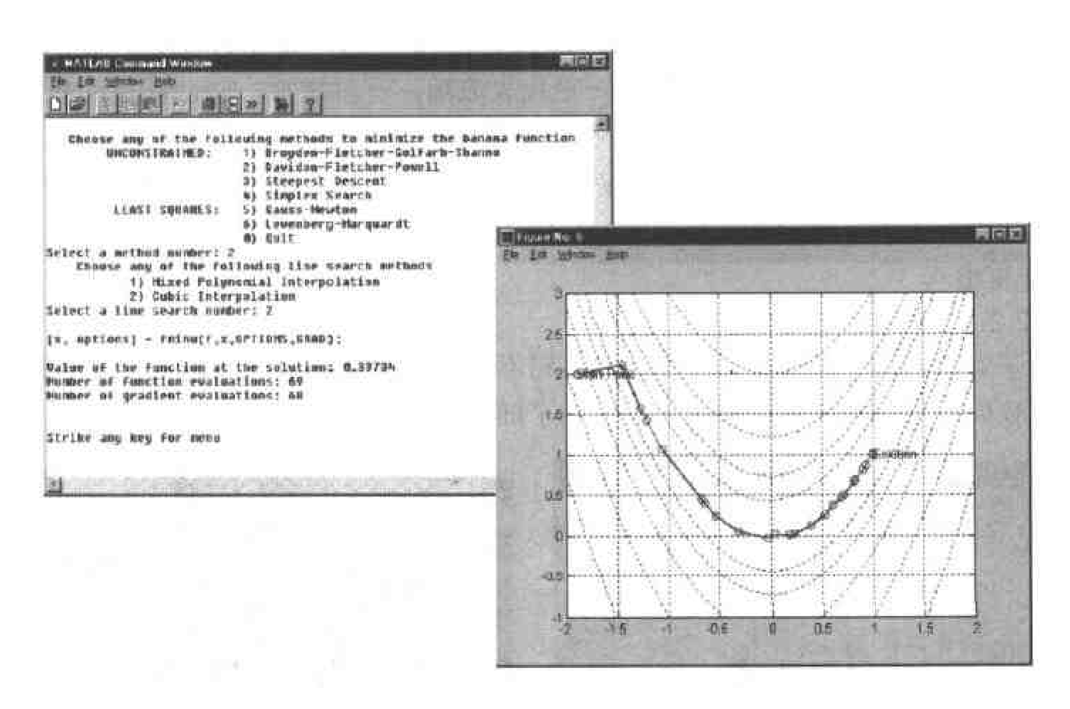


图 12.6　banana 优化算法

上述过程操作起来很麻烦，用户需要在一个窗口内完成各种算法的选择，在另一个窗口内观察算法的处理效果，这有悖于尽量缩小交互区域的原则。如果将这两个窗口集成在一起，使用起来的效果则截然不同。在 MATLAB 环境下运行下列指令：

bandem

运行结果如图 12.7 所示。

通过按钮，用户可以直接在该界面上选择算法，无需在两个界面上进行切换。这种设计加深了用户对各种算法运行效果的直观理解，增强了图形的可读性。

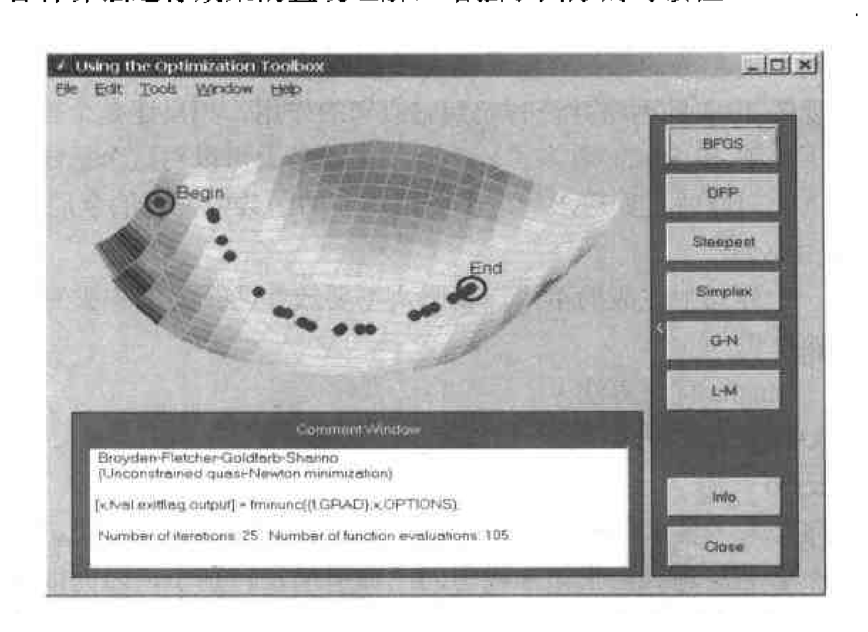


图 12.7　banana 优化算法界面

3) 用图形输入代替数值输入

例 12.2.3　三维形体观察视角的变换。

图 12.8 为三维形体观察界面，图中用环绕几何形体的两个圆圈来表示镜头和光源的轨迹，用户可以通过拖拉×改变镜头的位置，拖拉圆点改变光源的位置。

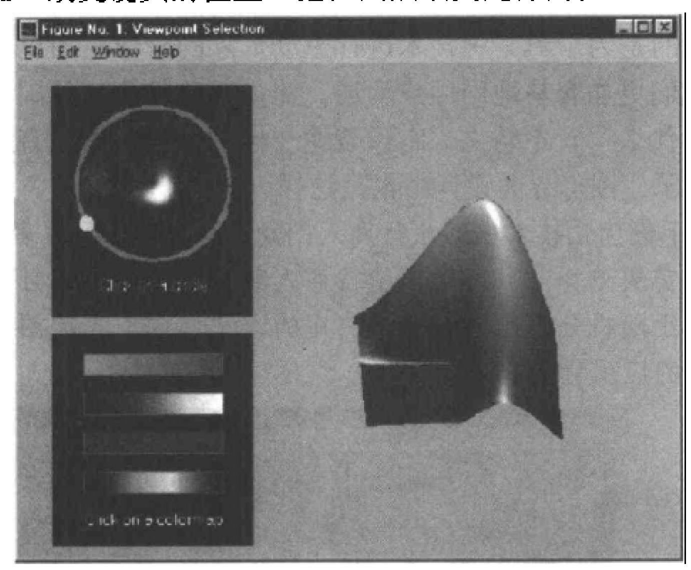


图 12.8　三维形体观察界面

对于用户来说，在图形界面下，图形输入比数值输入更加方便。为了从各个角度来观察一个三维几何形体，我们需要不断地进行坐标系的变换。用数值输入的方法，困难程度难以想象，其复杂性会让用户望而却步，而图形输入的方式则能很好地解决上述问题。

2. 一致性

一致性的含义很广泛，既包括操作指令的前后一致，也包括操作界面中各种图形对象摆放位置的一致性。这个原则基于以下认识，即用户的操作经验应有助于用户完成后面的操作；设计者不应该给用户太多的意外，否则会让用户一头雾水；前面执行这个命令是一种用法，后面却是另外一种用法，于是产生了歧义。优秀的界面设计往往注重这个问题，请看下面这个例子。

例 12.2.4　界面一致性比较(见图 12.9)。

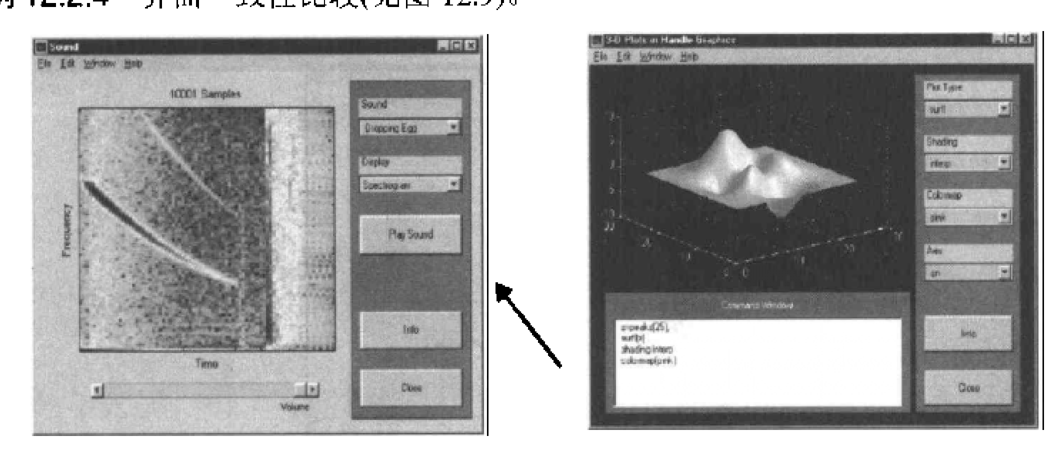


图 12.9　界面一致性比较

请注意在这两个界面中的 Info 和 Close 两个按钮的摆放位置及其作用。虽然这两个界面用来演示不同的技术，但是用户对其中任何一个界面的操作经历绝对有助于加深用户对另一个界面的了解。另外，还请读者注意界面中待处理图形和交互按钮的摆放位置。

3. 熟悉性

设计界面的目的是为了对某种技术进行生动形象的介绍。如果引入一个用户熟悉的东西，相信会使得我们更加容易地同用户沟通。比如，我们要向外国朋友介绍中华民族的图腾——龙，它长得什么样？不妨说，它长着蛇一样的身子，身上有鳞甲，就像鱼鳞……三言两语，就说清楚了。图形界面设计中也是这样。

图 12.10 是关于最优化技术中的旅行商人问题。商人需要到许多城市做生意，现在给出各个城市的位置，需要选择一条路线，使得商人到达各个城市并且总路程最短。实际上，城市是由一个平面上随机分布的一些点来决定的。我们以美国地图的轮廓为背景，很容易让用户熟悉该问题的应用背景。

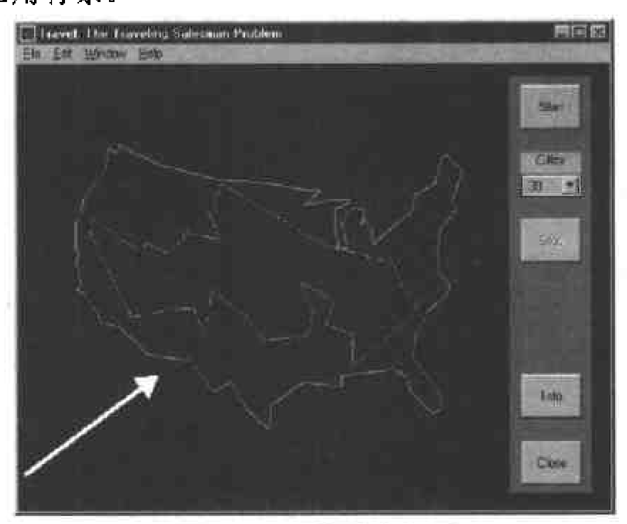


图 12.10 最短路径问题

下面再介绍一个例子。该例子也是从生活中我们所熟悉的事物出发的。

例 12.2.6 随机事件分布。

可以用抛掷硬币来解释随机数据的产生，如图 12.11 所示。左上图的那枚硬币与上例中的美国地图轮廓有异曲同工之妙。

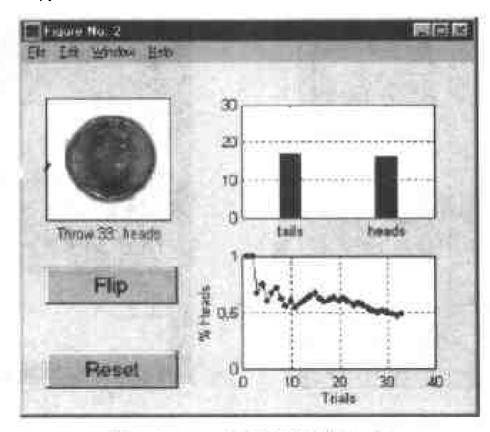


图 12.11 随机事件分布

12.2.2　动态界面的设计

目前我们的讨论主要偏向于静态界面的设计。当然，MATLAB 的 GUI 并不仅限于此。通过 GUI，设计者也可以开发出生动活泼的动态界面。在设计动态界面的时候，除了应遵循以上所强调的几点原则外，还要注重三点：即时性、连续性和可逆性。

例 12.2.7　傅立叶变换演示。

如图 12.12 所示，为了观察傅立叶变换中信号的频域和时域表示之间的对应关系，采用了动态界面的技术。通过拖拉滚动条或用鼠标直接拖拉波形，用户可直接完成波形频率和幅值的修改。

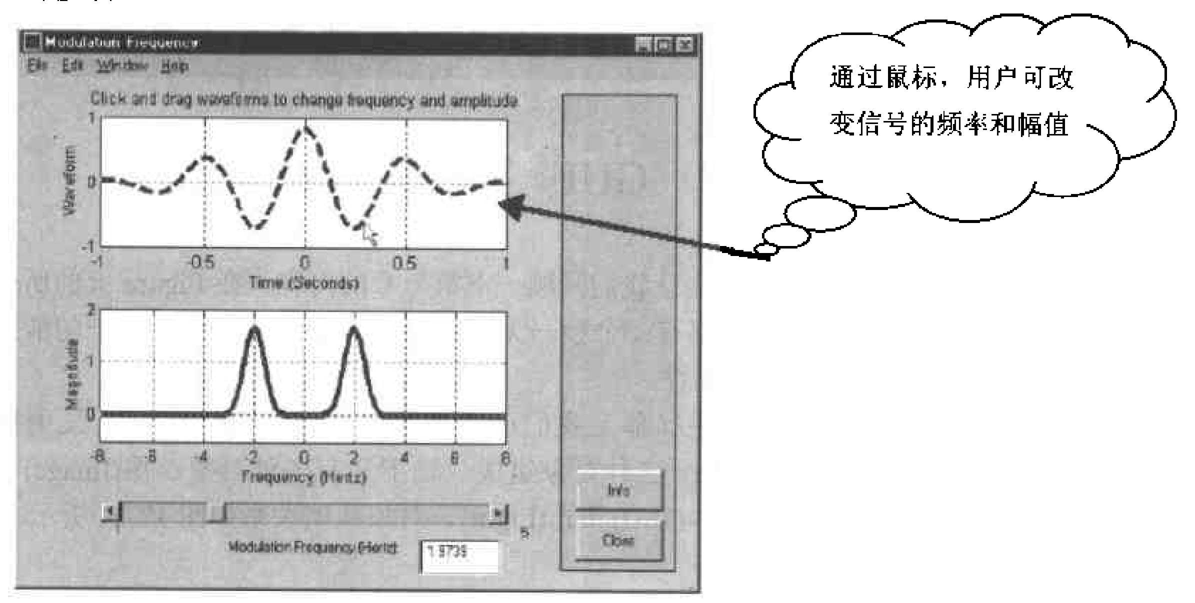


图 12.12　傅立叶变换演示

在设计动态界面的时候，设计者需要考虑计算量的大小。如果数据计算量太大，就会引起界面对用户输入响应的延时，达不到动态界面即时性与连续性的要求。这时，设计者必须考虑其它交互途径，比如用按钮来激发下一步操作。

12.2.3　开发流程

为了帮助读者对 MATLAB 的 GUI 开发过程有一个清晰的思路，图 12.13 给出了一个开发流程图。

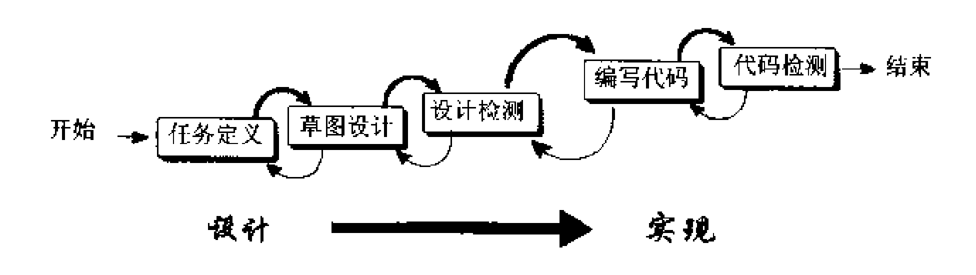


图 12.13　开发流程图

从图中可以看出，整个开发过程分为两个部分：设计阶段和实现阶段。一个好的设计是成功的一半，再一次提醒读者，要对设计予以足够的重视。很多人喜欢一下进入实现阶段，在他们看来，设计阶段解决不了什么问题，更愿意直接编写代码，他们的信念是“船到桥头自然直”。表面上这种想法无懈可击，实际上，在进行工程实践时，这种做法会带来很多问题，比如代码编写的随意性，会引发代码的可靠性问题，还有开发时间的延长，等等。现代工程开发的一大特征便是重视开发文档的建立。创造性的劳动都是在设计阶段进行的，开发文档便是对这些思想的记录。因此，请设计者不要忽视设计阶段。

注意：开发流程不是单向进行的，有时有一个反复的过程。后面会遇到一些无法解决的问题，因此需要对前面做局部的调整。这种反复过程是正常开发所无法避免的。严格按照上述流程进行开发带来的一个好处是，开发者能够从整体上把握设计，制定出合适的策略来解决问题。

12.3 GUI 实 现

在开始介绍编程实现 GUI 前，先让我们回顾一下第一章的内容。在 Figure 上的所有对象都是句柄图形的一个实例。它们都有一个独一无二的句柄。我们正是通过这个句柄把它们区分开来，从而改变每个对象的属性。

Root 可以看作是整个屏幕，在屏幕上我们可以放置多个 Figure。Figure 又由控件(Uicontrol)、轴(Axes)和菜单(Uimenu)三种对象组成。轴下又有多种对象：像(Image)、线(Line)、块(Patch)、面(Surface)、文本(Text)和光(Light)。对象从属关系如图 12.14 所示。

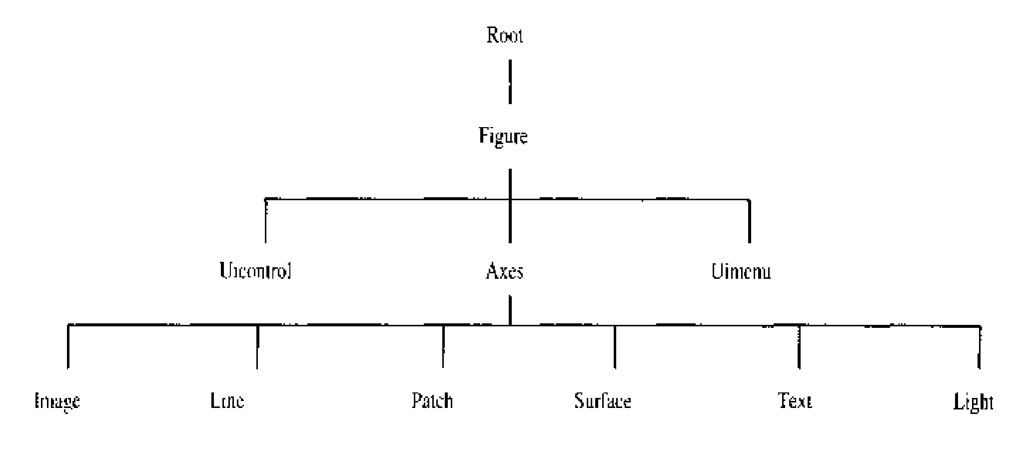


图 12.14　对象从属关系图

那么，如何获得这些对象的句柄？如何对各种对象中存在着的多种属性进行设置？本节首先详细介绍 GUIDE 的各种工具的使用，然后借助这些工具解决一些工程实例。希望这些工程实例能够为读者朋友的实际工作提供帮助。

本节的体系结构是这样的：第一个实例侧重于介绍 GUIDE 工具的使用，后面的部分从 MATLAB 的各种应用环境中，挑选出一些有代表性的工程实例。首先叙述实例的应用背景；接着介绍开发 GUI 用到的主要技术(为了帮助读者进一步了解代码结构，书中还对部分核心代码作了剖析，并加上了翔实的注释)；最后，列举了一些重要的函数命令，以备检索。

12.3.1　GUIDE 开发实例

1. 应用背景

用户通过按钮、列表框和可编辑文本框完成交互，实现对一个三维图形的控制，如图12.15 所示。

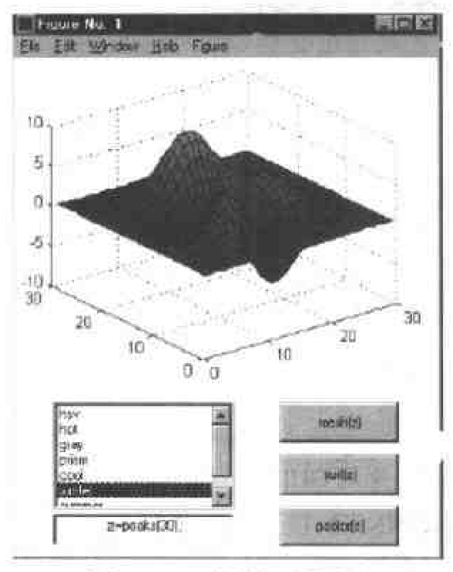


图 12.15　控制三维图形

2. 目的

该例的目的是介绍 GUIDE 创建图形界面的开发步骤，即如何利用控制面板(Control Panel)向图形上添加控件，如何利用属性编辑器(Property Editor)修改图形上各种对象的图形属性，如何利用回调函数编辑器(Callback Editor)编写回调函数来响应用户。这些都是最基本的操作。

3. 创建步骤

(1) 在 MATLAB 命令行中，运行指令 GUIDE，出现图 12.16 所示的窗口。

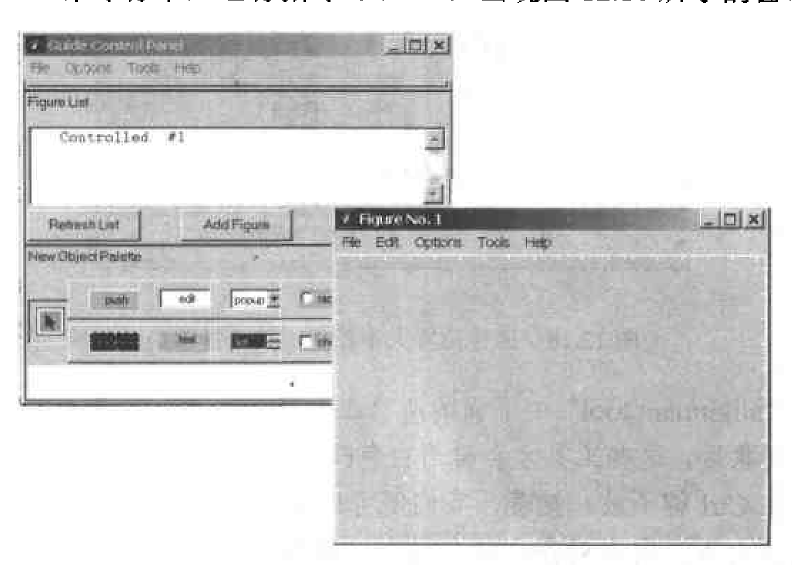


图 12.16　图形用户界面可视化设计环境

左上方是控制面板，右下方是一个新的图形(Figure)，我们所要做的工作都在这个图形上进行。

(2) 选中控制面板上新对象栏中的第二排第一个按钮——轴，在 Figure 上的合适区域画出坐标系。依同样的步骤，将按钮、表框、可编辑文本框加到 Figure 上，得到图 12.17 所示的的界面。

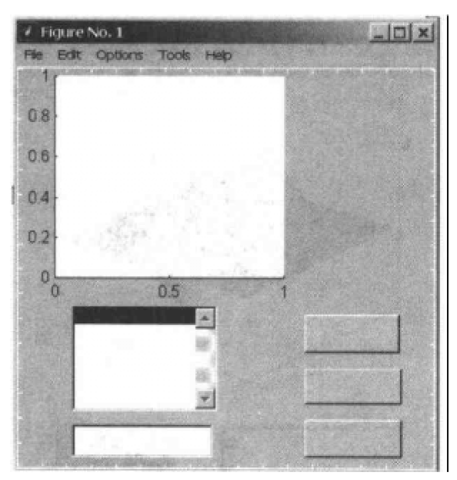


图 12.17 采用 GUIDE 工具设计的用户窗口界面

(3) 打开 GUIDE 中的 ALIGNMENT 工具，调整界面上的各个控件的间距，使得界面整齐划一。具体方法是，点击控制面板菜单中的“Tools”，从弹出菜单中选中“Alignment tool”，出现图 12.18 所示界面。

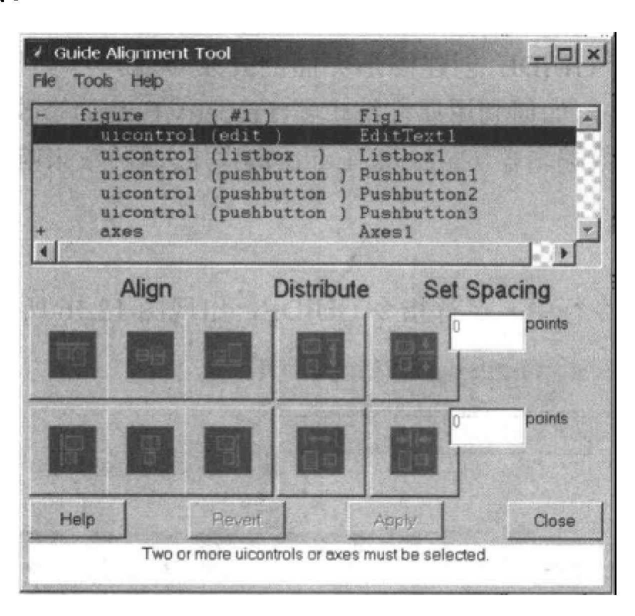


图 12.18 控件位置大小校正工具选择窗口

注意，目前“alignment tool”中下面的所有工具都是灰色，处于无效状态。只有从上面列表中选取多个对象后，这些工具才会转为白色(即有效状态)。选取多个对象的方法是：在选择对象时，按住 Ctrl 键不放。接着，我们便可以使用“Align”工具组中工具使得多个对象向下对齐，向上对齐，向左对齐，向右对齐，或者利用“Distribute”工具组中工具调整多个对象的上下间距，左右间距。

以上步骤已完成了界面的布置，下面需要向列表框中添加选项。

(4) 在 MATLAB 命令行中输入：

```
cmaps = {'hsv';'hot';'gray';'prism';'cool';'winter';'summer'};
```

于是，MATLAB 的工作空间中会生成一个字符串数组 cmaps。使用属性编辑器可以将该列表框的字符串属性修改为 cmaps。注意不要加引号！

(5) 打开回调函数编辑器，为列表框和按钮添加回调函数(见图 12.19)。先选中对象，接着选择函数类型“callback”，然后在编辑框中输入下列指令：

```
Value = get(gcbo,'Value');
%返回列表框的值
String = get(gcbo,'String');
%返回列表框的字符串
colormap(String{Value})
%根据列表框的值和字符串，改变 Figure 的色表
```

同样，也可以为按钮添加回调函数，代码如下：

```
CommandString = get(gcbo,'String');
%得到按钮所对应的画图指令
EditHandle = findobj(gcbf,'Tag','EditText1');
%得到GUIDE分配给可编辑框的句柄
ZString = get(EditHandle,'String');
%从编辑框中获得指令
eval(ZString);
%执行编辑框中的指令
eval(CommandString);
%执行按钮所对应的指令
```

现在 GUI 就可以工作了。在对它测试之前，记住要将其保存，否则将功亏一篑！

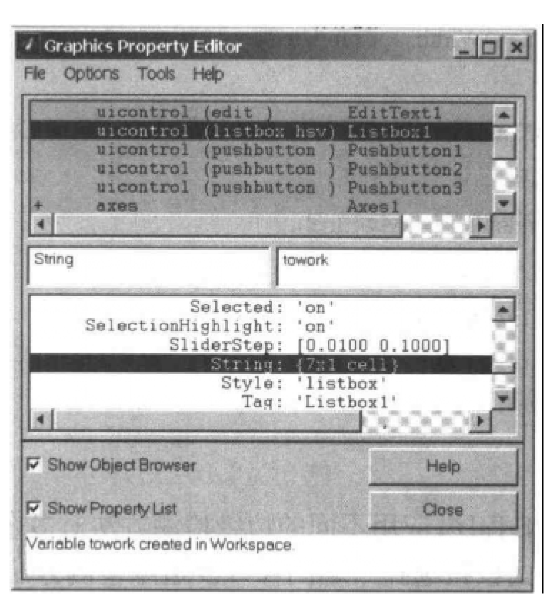


图 12.19　列表框控件添加选项窗口

4. 回调函数的编写

正如我们把整个开发过程分为设计和实现一样，我们也可以把实现分为界面的布置和回调函数的编写。GUIDE 很好地帮助我们完成了前一部分工作，剩下的便是回调函数的编写。如果注意以下几点，回调函数的运行效率会更高。

尽量使用短小的函数来实现回调，而不是一长串字符串；

采用如下的程序结构会增加程序的可读性(MATLAB 中的大部分示例程序都使用了这种结构)：

```
function mygui(action)
    switch(action)
    case 'load',
        load mydata;
        set(gcbf,'UserData',XYData)
    case 'plot'
        XYData=get(gcbf,'UserData')
        x=XYData(:,1);
        y=XYData(:,2);
        plot(x,y)
    case 'close'
        close(gcbf)
end
```

在回调函数内部，需要改变自身状态或其父对象的状态时，可以使用 gcbo 和 gcbf。

注意 findobj 指令，该指令可以帮助我们减少大量的冗余变量。比较下面两段代码：

(1) 代码一：

```
lineHandle = plot(rand(10,1))  %记住句柄，以备后面使用
... 大量代码 ...
set(lineHandle,'LineWidth',4,'Color','red')
```

(2) 代码二：

```
plot(rand(10,1),'Tag','My Line')  %不用记住代码，这是给它赋一个Tag
... 中间大量代码 ...
lineHandle = findobj('Tag','My Line');
...根据Tag找回所要操作对象的句柄
set(lineHandle,'LineWidth',4,'Color','red')
```

12.3.2 边缘检测实例

1. 应用背景

该实例通过调用EDGE程序，应用不同的边缘检测方法对多个图像进行处理。如图 12.20 所示，用户可以通过弹出式按钮选择一幅图像以及边缘检测方法，也可以设置边缘检测算法中的参数(参数集随着检测方法的不同而变动)。当完成设置后，用户可以通过“Apply”

按钮激活图像的处理，按照设定的方法和参数对图像进行处理 。

缺省状态下，EDGE 程序会自动选择适当的门限阈值，当然，为了满足用户对图像处理的要求，也可以手工设定门限阈值。

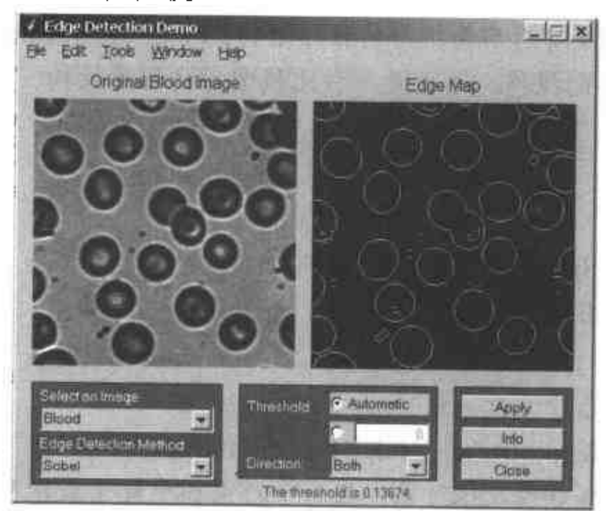


图 12.20　完整的边缘检测实例图形界面

2. 目的

使读者进一步熟悉各种控件的使用，并掌握复杂回调函数的编写。另外，该实例也介绍了如何根据用户的选择来动态更新界面的技术。

请读者注意，一个优美的界面是算法的漂亮外衣。如何给算法穿上这件外衣是我们设计者应该考虑的问题。毫无疑问，第一重要的是要量体裁衣。因此，界面是建立在算法基础之上的，算法对输入参数的要求应该转化为界面的语言，从而清晰地向用户表达。

源代码：

```
function edgedemo(action, varargin)
% 子函数说明:
% InitializeEDGEDEMO -        初始化图像，控件，轴
% ComputeEdgeMap -            调用 edge.m 计算原始图像的边缘图
% SelectMethod -              选取边缘检测算法，并是相应的控件有效或无效
% LoadNewImage -              读入选取的图像
% UpdateThreshCtrl -          从编辑框中获取门限值，并使“Apply”按钮有效
% UpdateDirectionality -      根据弹出菜单中的内容显示指示字符串
% Radio -                     设置 Radio Buttons 中的值，并使得门限编辑框有
                              效或无效
% UpdateLOGSize -             从编辑框中获取 LOG 过滤器的大小
% UpdateLOGSigma -            从编辑框中获取 LOG 过滤器的 Sigma 值
% ActivateSPRControls -       打开 Sobel, Prewitt, Roberts 方法中用到的控件
% ActivateLOGControls -       打开 LOG 方法中用到的控件

if nargin<1,
```

```
    action='InitializeEDGEDEMO';
end;
feval(action,varargin{:});
%注意，这里没有用前面推荐的程序结构。而是巧用 feval 函数来调用回调函数相
%应用户的界面操作，这也是一种比较常用的程序结构，往往用在回调函数比较
%复杂的情况下
return;

%  子函数  InitializeEDGEDEMO
function InitializeEDGEDEMO()

% 如果程序已经运行，则将它放到前台
h = findobj(allchild(0), 'tag', 'Edge Detection Demo');
            %0 代表根对象，在整个对象树中处于最顶层
if ~isempty(h)
    figure(h(1))
    return
end

screenD = get(0, 'ScreenDepth');
if screenD>8
    grayres=256;
else
    grayres=128;
end

    %下面的代码主要完成对界面对象的描述，包括大小、位置及其属性
    %由于篇幅所限，这里只给出 Figure 的设置
EdgeDemoFig = figure( ...
    'Name','Edge Detection Demo', ...
    'NumberTitle','off', 'HandleVisibility', 'on', ...
    'tag', 'Edge Detection Demo', ...
    'Visible','off', 'Resize', 'off',...
    'BusyAction','Queue','Interruptible','off', ...
    'Color', [.8 .8 .8], ...
    'IntegerHandle', 'off', ...
    'DoubleBuffer', 'on', ...
    'Colormap', gray(grayres));
```

```
figpos = get(EdgeDemoFig, 'position');
% 调整 Figure 窗口的大小
figpos(3:4) = [560 420];
horizDecorations = 10;  % resize controls, etc.
vertDecorations = 45;   % title bar, etc.
screenSize = get(0,'ScreenSize');
if (screenSize(3) <= 1)
    %找不到显示设备
    screenSize(3:4) = [100000 100000]; % don't use Inf because of vms
end
if (((figpos(3) + horizDecorations) > screenSize(3)) | ...
            ((figpos(4) + vertDecorations) > screenSize(4)))
    %屏幕太小无法正常运行该程序
    delete(EdgeDemoFig);
    error(['Screen resolution is too low ', ...
          '(or text fonts are too big) to run this demo']);
end
dx = screenSize(3) - figpos(1) - figpos(3) - horizDecorations;
dy = screenSize(4) - figpos(2) - figpos(4) - vertDecorations;
if (dx < 0)
    figpos(1) = max(5,figpos(1) + dx);
end
if (dy < 0)
    figpos(2) = max(5,figpos(2) + dy);
end
set(EdgeDemoFig, 'position', figpos);

rows = figpos(4); cols = figpos(3);
hs = (cols-512) / 3;         %设置水平间距
bot = rows-2*hs-256;         %图像的底部位置

%==================================
%本部分为控件和菜单的设置语句
%由于该部分程序比较简单，但篇幅较长，本书省略该部分内容。读者可以通过
%edit edgedemo 命令来阅读该程序
  ......
%==================================

set(EdgeDemoFig, 'Userdata', hdl, 'Visible', 'on');
```

```
%初始化中创建的所有对象的句柄都放入 hdl 中，然后又将 hdl 放入图形 Figure
%的'Userdata'。由此实现了句柄的传递。这是一种很好的共享数据的方法，请读
%者注意
drawnow
LoadNewImage(EdgeDemoFig);
drawnow
set(EdgeDemoFig, 'HandleVisibility', 'Callback');
set([hdl.Apply hdl.Help hdl.Close] , 'Enable', 'on');
return

%子函数  ComputeEdgeMap
function ComputeEdgeMap(DemoFig)

if nargin<1
    callb = 1;
    DemoFig = gcbf;
else
    callb = 0;
end

set(DemoFig,'Pointer','watch');
setstatus(DemoFig, 'Computing the edge map...');
hdl=get(DemoFig,'Userdata');
img = getimage(hdl.Image);

autothresh = get(hdl.RadioAutomatic, 'Value');

switch hdl.Method
case {'Sobel','Roberts','Prewitt'}
    if autothresh
        [edgemap,thresh]=edge(img,hdl.Method, dl.Directionality);
        setstatus(['The threshold is ' num2str(thresh) '.']);
    else
        edgemap = edge(img, ...
            hdl.Method, hdl.Threshold, hdl.Directionality);
        setstatus(DemoFig, '');
    end
case 'Laplacian of Gaussian'
    if autothresh
```

```
        [edgemap,thresh] = edge(img, 'log', [], hdl.LogSigma);
        setstatus(DemoFig, ['The threshold is ' num2str(thresh) '.']);
    else
        edgemap = edge(img, 'log', hdl.Threshold, hdl.LogSigma);
        setstatus(DemoFig, '');
    end
case 'Canny'
    if autothresh
        [edgemap,thresh] = edge(img, 'canny', [], hdl.LogSigma);
        setstatus(DemoFig, ['High threshold is ' num2str(thresh(2)) '.']);
    else
        [edgemap,thresh] = edge(img, 'canny', hdl.Threshold, hdl.LogSigma);
        setstatus(DemoFig, '');
    end
otherwise
    error('EDGEDEMO: Invalid edge detection method.');
end

set(hdl.Edge, 'CData', edgemap);
set(hdl.Apply, 'Enable', 'off');
set(DemoFig,'Pointer','arrow');
drawnow

%子函数  SelectMethod
function SelectMethod

DemoFig = gcbf;

hdl = get(DemoFig, 'userdata');
v = get(hdl.MethodPop,{'value','String'});
hdl.Method = deblank(v{2}(v{1},:));
switch hdl.Method
case {'Sobel','Prewitt'}
    ActivateSPRControls(DemoFig);
    set(hdl.sprDirPop, 'Enable', 'on');
case 'Laplacian of Gaussian'
    ActivateLOGControls(DemoFig);
    set(hdl.logSigmaCtrl, 'String', '2');
    hdl.LogSigma = 2;
```

```
case 'Canny'
    ActivateLOGControls(DemoFig);
    set(hdl.logSigmaCtrl, 'String', '1');
    hdl.LogSigma = 1;
case 'Roberts'
    ActivateSPRControls(DemoFig);
    set(hdl.sprDirPop, 'Enable', 'off', 'value', 1);
otherwise
    error('EDGEDEMO: invalid method specifier.');
end

set(hdl.Apply, 'Enable', 'on');
set(DemoFig, 'userdata', hdl);
setstatus(DemoFig, ['Press "Apply" to compute edges.']);

%子函数   LoadNewImage
function LoadNewImage(DemoFig)

if nargin<1
    callb = 1;      %程序由回调函数调用
    DemoFig = gcbf;
else
    callb = 0;      %程序处于初始化过程
end

set(DemoFig,'Pointer','watch');
%由于图像加载时间可能比较长，在加载前把鼠标的形状改成沙漏形
hdl=get(DemoFig,'Userdata');
v = get(hdl.ImgPop,{'value','String'});
name = deblank(v{2}(v{1},:));
setstatus(DemoFig, ['Loading the ' name ' image...']);
drawnow

switch name
case 'Aluminum',
    alumgrns2 = [];                         % Parser hint
    load imdemos alumgrns2
%注意，这里图像被放在工作空间 imdemos 中，所以使用了 load; 否则，需要使
%用 imread 的函数。具体使用请参照前面的介绍
```

```
        img = alumgrns2;
    case 'Blood',
        blood2 = [];
        load imdemos blood2
        img = blood2;
    case 'Rice',
        rice3 = [];
        load imdemos rice3
        img = rice3;
    case 'Saturn',
        saturn2 = [];
        load imdemos saturn2
        img = saturn2;
    case 'Eight Bit',
        eight = [];
        load imdemos eight
        img = eight;
    case 'Circuit',
        circuit4 = [];
        load imdemos circuit4
        img = circuit4;
    case 'Vertigo',
        vertigo2 = [];
        load imdemos vertigo2
        img = vertigo2;
    case 'Bone Marrow',
        bonemarr2 = [];
        load imdemos bonemarr2
        img = bonemarr2;
    otherwise
        error('EDGEDEMO: Unknown Image Option!');
    end

    set(hdl.Image, 'Cdata', img);
    set(get(hdl.ImageAxes,'title'),'string',['Original ' name ' Image']);
    set(DemoFig,'Pointer','arrow');
    if callb
        set(hdl.Apply, 'Enable', 'on');
    end
```

```
drawnow
if ~strcmp(hdl.Method, 'Laplacian of Gaussian')
    if get(hdl.RadioAutomatic, 'Value')==0
        Radio('auto',DemoFig);
    end
end
ComputeEdgeMap(DemoFig);
return;

%子函数  UpdateThreshCtrl
function UpdateSprThresh()
DemoFig = gcbf;
hdl = get(DemoFig, 'UserData');
v = hdl.Threshold;
s = get(hdl.ThreshCtrl,'String');
vv = real(evalin('base',['[' s ']'],num2str(v)));
%检验阈值是否输入
if isempty(vv) | ~isreal(vv) | vv(1)<0
    vv = v;
    set(gcbo,'String',num2str(vv));
    return
end
vv = round(vv(1)*1000000)/1000000;
set(gcbo,'String',num2str(vv));
hdl.Threshold = vv;
set(hdl.Apply, 'Enable', 'on');
setstatus(DemoFig, 'Press "Apply" to compute edges.');
set(DemoFig, 'UserData', hdl);
return

%子函数  UpdateDirectionality
function UpdateDirectionality()

DemoFig = gcbf;
hdl = get(DemoFig, 'UserData');
v = get(hdl.sprDirPop,{'value','String'});
dir = deblank(v{2}(v{1},:));
set(hdl.sprDirPop, 'userdata', dir);
hdl.Directionality = dir;
```

```
set(hdl.Apply, 'Enable', 'on');
setstatus(DemoFig, 'Press "Apply" to compute edges.');
set(DemoFig, 'UserData', hdl);
return

%子函数  Radio
unction Radio(control, DemoFig)

if nargin<2
    DemoFig = gcbf;
end
hdl = get(DemoFig, 'UserData');
if strcmp(control, 'auto')
    set(hdl.RadioAutomatic, 'Value', 1);
    set(hdl.RadioManual, 'Value', 0);
    set(hdl.ThreshCtrl, 'Enable', 'off');
    set(hdl.Apply, 'Enable', 'on');
    setstatus(DemoFig, 'Press "Apply" to compute edges.');
elseif strcmp(control, 'manual')
    set(hdl.RadioAutomatic, 'Value', 0);
    set(hdl.RadioManual, 'Value', 1);
    set(hdl.ThreshCtrl, 'Enable', 'on');
    set(hdl.Apply, 'Enable', 'on');
    setstatus(DemoFig, 'Press "Apply" to compute edges.');
end
return

%子函数  UpdateLOGSigma
%当用户输入 SIGMA 值后，一方面需要验证取值的有效性，另一方面需要更新
%界面
function UpdateLOGSigma()
DemoFig = gcbf;
hdl = get(DemoFig, 'UserData');
v = hdl.LogSigma;
s = get(hdl.logSigmaCtrl,'String');
vv = real(evalin('base',s,num2str(v)));
if isempty(vv) | ~isreal(vv) | vv(1)<0
%验证输入的 SIGMA 值是否有效
    vv = v;
```

```
    set(hdl.logSigmaCtrl,'String',num2str(vv));
    return
end
vv = round(vv(1)*100)/100;
set(hdl.logSigmaCtrl,'String',num2str(vv));
hdl.LogSigma = vv;
set(hdl.Apply, 'Enable', 'on');
setstatus(DemoFig, 'Press ''Apply'' to compute edges.');
set(DemoFig, 'UserData', hdl);
return

%下面两个子程序都是通过改变控件的可见属性来完成界面的切换的

%子函数  ActivateSPRControls
%该子程序激活 SPR 三种算法使用的公共控件
function ActivateSPRControls(DemoFig)
hdl = get(DemoFig, 'UserData');
set([hdl.sprDirPop hdl.sprDirLbl], 'Visible', 'on');
set([hdl.logSigmaCtrl hdl.logSigmaLbl], 'Visible', 'off');

%子函数  ActivateLOGControls
%该子程序激活 LOG 算法中使用的控件
function ActivateLOGControls(DemoFig)
hdl = get(DemoFig, 'UserData');
set([hdl.logSigmaCtrl hdl.logSigmaLbl],'Visible', 'on');
set([hdl.sprDirPop hdl.sprDirLbl], 'Visible', 'off');
```

3. 小技巧

在该实例中，用户需注意以下两点：

(1) 使用 hdl 结构体来保存图形界面中各种对象的句柄。这些句柄在创建图形对象时产生，并在以后的回调函数中多次用到。

(2) 各个对象中的“userdata”属性域的妙用。我们可以通过它在子程序之间传递数据。这样做避免了全局变量的使用。

参考文献

[1] 章毓晋. 图像处理和分析. 北京：清华大学出版社，1999
[2] 周新伦，柳健. 数字图像处理. 北京：国防工业出版社，1986
[3] MATLAB 6.1 帮助文档. MathWorks, Inc. 2001
[4] 孙家广，杨长贵. 计算机图形学. 北京：清华大学出版社，1995
[5] Kenneth R Castle. 数字图像处理. 北京：电子工业出版社，1998
[6] ROBERT L IBBEY. 信号与图像处理. 北京：电子工业出版社，1997
[7] 陈桂明，张明照，戚红雨编著. 应用 MATLAB 语言处理数字信号与数字图像. 北京：科学出版社，2000
[8] 楼顺天，于卫，闫华梁编著. MATLAB 程序设计语言. 西安：西安电子科技大学出版社，1997
[9] 崔屹. 数字图像处理技术与应用. 北京：电子工业出版社，1997
[10] 吕风军. 数字图像处理编程入门. 北京：清华大学出版社，1999